FORMAL SPECIFICATION OF SOFTWARE ARCHITECTURE DESIGN TACTICS FOR THE SECURITY QUALITY ATTRIBUTE

Andrew Michael Wyeth
B.S., University of Victoria, Victoria, Canada, 2002

PROJECT

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

SPRING
2009
FORMAL SPECIFICATION OF SOFTWARE ARCHITECTURE DESIGN TACTICS FOR THE SECURITY QUALITY ATTRIBUTE

A Project

by

Andrew Michael Wyeth

Approved by:

Dr. Cui Zhang, Committee Chair

Date: 5/13/2009

Dr. William Mitchell, Second Reader

Date: May 13, 09
Student: Andrew Michael Wyeth

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

Dr. Cui Zhang, Graduate Coordinator

5/13/2009

Date

Department of Computer Science
Abstract

of

FORMAL SPECIFICATION OF SOFTWARE ARCHITECTURE DESIGN TACTICS FOR THE SECURITY QUALITY ATTRIBUTE

by

Andrew Michael Wyeth

Designing the software architecture of a system is an important step in creating a system that will meet both the functional and non-functional requirements. Bass, Clements and Kazman in “Software Architecture in Practice” propose a method to use design tactics to create the Software Architecture of a system. For each quality attribute a set of design tactics were developed. Security is one such quality attribute.

The security requirements of a system must be taken into account from the start. Adding security adversely affects the other quality attributes so it is difficult to add security to a product after it has been designed. When defining the security tactics of a system, there is currently no way to formally prove the implementation of the tactics. There is a semantic gap between the software architecture, high level design and implementation of a system. One way to bridge this gap is to use formal specifications. The formal specifications can be used as a template when designing new systems and to analyze the architecture more rigorously.

This project provides a Z specification for the Software Architectural Tactics for the Security Quality Attribute. A model of a system is created and each tactic is defined with respect to the model. Each tactic is independent however, the system encompasses all the required functionality for all the tactics.
ACKNOWLEDGMENTS

I would like to thank my wife, Greta and my son Rowan for all their love and support in helping me to complete this project. I would also like to thank my advisor, Dr. Cui Zhang for all of her guidance and support and finally, I would like to thank my second reader, Dr. William Mitchell for all his help.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Acknowledgments</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
</tbody>
</table>

## Chapter

1. **INTRODUCTION** .................................................................................................................. 1
2. **BACKGROUND AND RELATED WORK** ........................................................................... 3
3. **FORMAL SPECIFICATION OF THE DESIGN TACTICS** .................................................. 9
   3.1 Construction of Tactic Specification ............................................................................ 9
   3.2 Base System Model Specification .............................................................................. 10
   3.3 Authentication Tactic Model Specification ............................................................ 14
   3.4 Authorization Tactic Model Specification .............................................................. 19
   3.5 Data Confidentiality Tactic Model Specification ..................................................... 25
   3.6 Data Integrity Tactic Model Specification .................................................................. 29
   3.7 Auditing Tactic Model Specification ......................................................................... 33
   3.8 Intrusion Detection Tactic Model Specification ....................................................... 40
   3.9 Limit Access, Limit Exposure and Availability Tactics ........................................... 44
   3.10 Combined Authorization and Authentication Model Specification ..................... 46
4. **DISCUSSION** .................................................................................................................. 50
   4.1 Conclusion ................................................................................................................... 50
   4.2 Applications and Uses ............................................................................................... 50
   4.3 Future Work .................................................................................................................. 52

## Appendix

A. Base System Model Specification .................................................................................... 55
B. Authentication Tactic System Model Specification .......................................................... 63
C. Authorization Tactic System Model Specification ............................................................ 74
D. Data Confidentiality Tactic System Model Specification ............................................... 99
E. Data Integrity Tactic System Model Specification .......................................................... 108
F. Auditing Tactic System Model Specification .................................................................. 118
G. Intrusion Detection Tactic System Model Specification .................................................. 133
H. Combined Authorization and Authentication System Model Specification .................. 150

References .......................................................................................................................... 183
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Figure 1. Construction of Tactic Specification</td>
<td>10</td>
</tr>
<tr>
<td>2.</td>
<td>Figure 2. Base System Specification</td>
<td>11</td>
</tr>
<tr>
<td>3.</td>
<td>Figure 3. Process Specification</td>
<td>12</td>
</tr>
<tr>
<td>4.</td>
<td>Figure 4. Add User Specification</td>
<td>13</td>
</tr>
<tr>
<td>5.</td>
<td>Figure 5. Change File Specification</td>
<td>14</td>
</tr>
<tr>
<td>6.</td>
<td>Figure 6. Revised System Specification to Include Authentication</td>
<td>15</td>
</tr>
<tr>
<td>7.</td>
<td>Figure 7. Authenticate User Specification</td>
<td>16</td>
</tr>
<tr>
<td>8.</td>
<td>Figure 8. Add Authentication Specification</td>
<td>17</td>
</tr>
<tr>
<td>9.</td>
<td>Figure 9. Revised Add User and Change File Specifications to Include Authentication</td>
<td>18</td>
</tr>
<tr>
<td>10.</td>
<td>Figure 10. Revised System Specification to Include Authorization</td>
<td>20</td>
</tr>
<tr>
<td>11.</td>
<td>Figure 11. Add User ACM Specification</td>
<td>22</td>
</tr>
<tr>
<td>12.</td>
<td>Figure 12. Revised Add User Specification to Include Authorization</td>
<td>23</td>
</tr>
<tr>
<td>13.</td>
<td>Figure 13. Revised Start Process Specification to Include Authorization</td>
<td>24</td>
</tr>
<tr>
<td>14.</td>
<td>Figure 14. Revised System Specification to Include Data Confidentiality</td>
<td>26</td>
</tr>
<tr>
<td>15.</td>
<td>Figure 15. Revised Read File Specification to Include Data Confidentiality</td>
<td>27</td>
</tr>
<tr>
<td>16.</td>
<td>Figure 16. Revised Add File Specification to Allow Multiple Secrets per Filename</td>
<td>27</td>
</tr>
<tr>
<td>17.</td>
<td>Figure 17. Revised Change File Specification to Include Data Confidentiality</td>
<td>28</td>
</tr>
<tr>
<td>18.</td>
<td>Figure 18. Revised File Properties Specification to Include Data Integrity</td>
<td>30</td>
</tr>
<tr>
<td>19.</td>
<td>Figure 19. Check Integrity Specification</td>
<td>31</td>
</tr>
<tr>
<td>20.</td>
<td>Figure 20. Revised Read File Specification to Check Data Integrity</td>
<td>32</td>
</tr>
<tr>
<td>21.</td>
<td>Figure 21. Revised System Specification to Include Auditing</td>
<td>34</td>
</tr>
<tr>
<td>22.</td>
<td>Figure 22. Log Element Specifications</td>
<td>35</td>
</tr>
<tr>
<td>23.</td>
<td>Figure 23. Change in File and Process Properties Function Specifications</td>
<td>36</td>
</tr>
<tr>
<td>24.</td>
<td>Figure 24. Revised Read File Specification to Include Auditing Using Lets</td>
<td>37</td>
</tr>
<tr>
<td>25.</td>
<td>Figure 25. Revised Read File Specification to Include Auditing with Local Variables in the Schema Declaration</td>
<td>38</td>
</tr>
<tr>
<td>26.</td>
<td>Figure 26. Revised StopProcess_ProcessIDNotExists Specification to Include Auditing</td>
<td>39</td>
</tr>
<tr>
<td>27.</td>
<td>Figure 27. Revised System Specification to Include Intrusion Detection</td>
<td>41</td>
</tr>
<tr>
<td>28.</td>
<td>Figure 28. IDS_Alert, IDS_NoAlert and IDS_System Specification</td>
<td>42</td>
</tr>
</tbody>
</table>
29. Figure 29. Revised Add User Specification to Include Intrusion Detection ............ 43
30. Figure 30. Combined Authorization and Authentication System Specification .......... 47
31. Figure 31. AddAuthenticationACM Specification ......................................................... 48
Chapter 1

INTRODUCTION

An important step in creating a system that meets the users functional and non-functional requirements is the design of the software architecture. During this phase of development, special attention must be paid to the non-functional requirements. Bass, Clements and Kazman [2] create that emphasis through the use of quality attributes, those requirements on the system that dictate the software architecture of the system. To achieve those qualities, the authors recommend the use of software architecture design tactics. Software architecture design tactics are high level design decisions. Each design tactic will satisfy one or more quality attributes and may adversely affect others [2].

Security is one set of quality attributes. As most applications work in an online environment, they must be secure to prevent compromise of the entire system [2]. There are three classes of tactics for security. The first class is Resisting Attacks which consists of: Authenticate Users, Authorize Users, Maintain Data Confidentiality, Maintain Data Integrity, Limit Exposure, and Limit Access. Second, Detecting Attacks which consists of using Intrusion Detection Systems. Lastly, Recovering From Attacks which consists of Auditing and Restoration. However, when defining the security tactics of a system, there is currently no way to formally prove the implementation of the tactics. One way to establish confidence in the implementation of the tactics is to use formal methods to define the system. The formal definition can be used to show that it achieves the tactics [2].
Formal methods are mathematical logic based procedures used to precisely define system requirements. These definitions enable us to rigorously analyze the system to prove certain properties, such as the confidentiality of the systems data. To accomplish these procedures, formal languages are used. A formal language is a language based on a mathematical model that can be used in formal methods. One such language is Z which is based on typed set theory [5]. Z is a language specifically designed to describe computer system models and as such is a very concise and powerful way to describe the high level constructs of a system.

The formal analysis to prove a system correctly implements the various security tactics has two parts. First, the security tactics need to be formally defined. Second, the proposed system must be formally defined and then shown to be equivalent to the security tactic definition. This project will define the security tactics with respect to a generic system.

The Software Architectural Security Tactics need to be defined with respect to a system so the first step in this project is to define the basic but generic system model. Then, each tactic will be defined in natural language and then formally as an extension to the basic system. The new extended system will be proven to be consistent with the original system. Finally, all the tactics will be combined to show the formal specification of a system model that has all the security quality attributes.
Chapter 2

BACKGROUND AND RELATED WORKS

In 1996 Shaw and Garlan in their book “Software Architecture Perspectives on an Emerging Discipline” show the feasibility and usefulness of writing formal specifications for different aspects of Software Architectural components or patterns [12]. This formalization is an important step in turning Software Architecture into an engineering discipline. Formal specifications allow the architecture to be rigorously modeled, analyzed and reused. Modeling is important to determine the feasibility of the system or to determine which of several models is the best option. Analysis of the formal specifications leads to a deeper understanding of the foundations of Software Architecture [12]. This analysis will lead to a “theory of software architecture” [12]. Also, having formal specifications of various software architectures allows for easy reuse; the template of the architecture can be used as a starting point for the design [12].

Shaw and Garlan in [12] demonstrate the above principles through three examples written in Z. First, they formally define the architecture of an oscilloscope, a tool for analyzing wave patterns and show how it implements the pipes and filters architectural style. This shows how a simple architecture can be written concisely and precisely and at the right level of abstraction.

Second, they define the pipes and filters architectural style and show how building on it would be easier than defining the oscilloscope from scratch. Also, in defining the pipes
and filters architectural style Shaw and Garlan [12] emphasize how an architectural style needs to be general and abstract enough to capture the essence of the style without enforcing too many constraints. They also show how easily a refinement to an architectural style can be defined. By adding a single constraint to the system they convert the pipes and filters architectural style to the pipeline style (where each filter is connected to only one input and one output pipe).

Finally, Shaw and Garlan [12] define the implicit invocation architectural style. The formalization of implicit invocation emphasized the need for the correct level of abstraction in the architecture. Specifically, it is not defined whether the events generated in the implicit invocation architectural style can contain data but, depending on the system being created, it might be necessary for the events to carry data. When the style is refined, one of the refinements will be to define exactly what an event is.

Since Shaw and Garlan published their book, significant advances have been made in Software Architectural Methodologies. During the architectural design of a system, the focus is on achieving the required quality attributes. Quality attributes are the properties of a system that do not affect what it does, only how it performs its task. Security, modifiability, usability, performance and reliability are all quality attributes [2]. The software architecture is essential to obtaining the quality attributes for any given system because it is the first place in the design cycle that quality attributes can be addressed.
The quality attributes that a system needs are defined through scenarios. A scenario consists of a source, a stimulus, an artifact, a response and a response measure [2].

The source is the initiator of the request, the stimulus is the request itself, the artifact is the portion of the system that is accessed, the response is what the system should do and the response measure is how to determine to what degree the system achieves the quality attribute. Once the quality attribute requirements have been defined for the system through the scenarios, they can be achieved through the use of software architectural tactics [2].

Software architectural tactics are design decisions [2]. Each tactic affects one or more quality attributes. For example, adding confidentiality to the system's data adds to the security of the system but reduces performance as the data must be decrypted before use. The selection of the architectural tactics for a system will help determine its architecture as the tactics will constrain what architectures are possible [2].

Security, as previously stated, is a system quality. There are various tactics to achieve the different types of security that systems require. According to [2], the tactics can be categorized based on whether they resist attacks, detect attacks or recover from attacks. The tactics that resist attacks are: authenticate users, authorize users, maintain data confidentiality, maintain integrity, limit exposure and limit access. The first four tactics correspond to their associated intuitive definitions. The idea of limit exposure is to
provide only a few services on each host, like not allowing the President and Vice-President of a company to fly on the same jet. The idea of limit access is to restrict what external entities can access the system, like a lock for a door.

While formal methods have been associated with software architecture since the mid-nineties, formal methods have been associated with security for even longer. Since computer security became an issue, people have looked at formal models of the security of a system. This is in part because without a formal definition of security, there is no way to precisely determine if a system is secure or not [6].

Network security is one area that has seen a significant amount of research into formalizing and analyzing its security properties. There are two complementary approaches to analyzing security protocols, determining the protocol requirements and proving the correctness of an existing protocol [4] [7]. CSP (Communicating Sequential Processes) is used to define authentication and confidentiality, two aspects of security, over a network. In doing so they explore the various types of authentication that are available for communication such as sender authentication and intent to send a message [4]. Z is used to define the Needham-Schroeder Public Key Protocol and prove it is insecure. This work is based on a flaw in the protocol. The Needham-Schroeder Public Key Protocol is then fixed and proved correct [7].
Another avenue of research is specifying the security components required to meet the Common Criteria. The Common Criteria is an international standard security evaluation system which defines several levels of security that systems can achieve [3]. The Common Criteria consists of 251 different security requirements written in natural language [8]. Morimoto et al. [8] have written Z templates for all 251 elements. This enables people to use the templates and makes it easier to achieve the desired Common Criteria security level.

Specifying the properties and models for aspects systems rather than for the whole system is another approach. Abdallah and Khayat [1] create a Z specification for role based access control models (RBAC). RBAC is a model that controls access based on what role or roles the user is currently employing.

Song et al. [13] create a system model and prove that an intrusion detection system can detect unauthorized access to the password file. Using ACL2 (a formal language), a unix like system model is defined and logging is explicitly added. Then the intrusion detection systems detection rules are defined and they are proved to be correct. The approach this paper uses is similar to the approach Song et al. took in proving the correctness of their system.

Z is a formal language based on typed set theory [5]. It is a good language to describe software architectures because it “allows a concise expression of the functional properties
of computer systems" [5]. There are also several powerful tools which work with Z.

There are programs that convert Object Z to Java or Spec # [9] [11] [10] and a tool which allows integration of Z specifications into Microsoft Word documents [16].
Chapter 3

FORMAL SPECIFICATION OF THE DESIGN TACTICS

3.1 Construction of Tactic Specification

The specification for each design tactic is built upon a simple base system model which is explained in Section 3.2. Each tactic's model except intrusion detection is built upon the base system model. Intrusion detection is built upon the auditing system model. The operations for each design tactic build upon the extended system model and modify the base system operations. Figure 1 shows how the tactics will be constructed. Starting with the base system model, the base system operations are specified and then each tactic is built upon the model and operations below it. The integrity tactic does not extend the base system model, it only refines the specification.
There is extensive use of the Z formal language, for a good introduction to Z see [5] and [14].

3.2 Base System Model Specification

This section presents the base system model specification. The full specification is in Appendix A.

The base system is a very simple, high level definition of a software system. The system is described in terms of an operating system as this makes it easier to understand and provide a ready source of examples. The specification can be applied to applications or
other systems by a simple mapping. The base system consists of three elements: a
userlist, a filelist, and a processlist as shown in Figure 2. USER, FILENAME, and
FILE_PROPERTIES are undefined sets, they depend on the system to be implemented.

\[
\text{SYSTEM} \\
\text{filelist} : \text{FILENAME} \mapsto \text{FILE_PROPERTIES} \\
\text{userlist} : \mathcal{P} \text{ USER} \\
\text{processlist} : \mathcal{P} \text{ PROCESS} \\
\forall p_1, p_2 : \text{processlist} \bullet p_1.\text{process_id} = p_2.\text{process_id} \iff p_1 = p_2 \\
\forall p : \text{processlist} \bullet p.\text{filename} \in \text{dom filelist} \land p.\text{owner} \in \text{userlist}
\]

Figure 2. Base System Specification

The filelist is a partial function from a filename to a file property. Thus, each filename is
unique in the system and filenames do not have to map to a file property.

The basis of the process is that it is a file run by a user. Figure 3 shows the process
specification, it consists of the user running the process, the file being run, the current
state of the process and the processes id. PROCESS_ID and PROCESS_STATE are
undefined sets. The PROCESS_STATE represents the current state of the process. As
specified in Figure 2, the system requires the process_id is unique and that the file being
executed and the user running the process both exist. This model does not consider the
sharing of resources, each operation is considered atomic. If concurrency is required then
the model would need to be extended to include a concurrency mechanism.
There are many operations on even this simple system: adding and deleting users; adding, deleting, reading, and changing files; starting, stopping and changing processes. Two representative operations are discussed here: AddUser and ChangeFile.

The AddUser operation, shown in Figure 4, consists of two possible cases: AddUser_OK - a user is successfully added to the set of users and AddUser_Exists - the user already exists so an error is returned.
AddUser_OK

\[\Delta SYSTEM \]
newUser? : USER
r! : Response

newUser? \in userlist
userlist' = userlist \cup \{newUser?\}
r! = OK

AddUser_Exists

\[\Xi SYSTEM \]
newUser? : USER
r! : Response

newUser? \in userlist
r! = UserExists

AddUser \equiv AddUser_OK \lor AddUser_Exists

Figure 4—Add User Specification

ChangeFile, shown in Figure 5, is slightly more complicated but still consists of two cases: ChangeFile_OK, where the filelist mapping is changed to reflect the new file properties (A \oplus B is the relational override operation where the mappings in A are overwritten by the mappings in B.) ChangeFile_NotExists returns an error because the file does not exist.
This specification describes the parts of a system that are monitored or extended to provide the various security tactics. The system as defined here has no security properties whatsoever, so any entity can manipulate the system.

3.3 Authentication Tactic Model Specification

This section presents the authentication tactic model specification. The full specification is in Appendix B.
Authentication is the first security tactic to be added to the base system. Authentication is the act of verifying a user's identity [3]. In software systems, authentication is achieved by having the user provide their name and a piece of information that uniquely identifies them [3]. This unique information is provided through the AUTHENTICATION_CONTROL set. Thus, the base system is extended to include a partial function from USER to an AUTHENTICATION_CONTROL set as shown in Figure 6. Note that each user can have any number of authentication controls and thus there can be users in the userlist that are not authenticated. This means the creation of a user is a two-step process, first the user is created, then the user is mapped to an authentication control. This allows the system to provide separation of concerns, which is one of the secure design principles [3].

```
SYSTEM
filelist : FILENAME → FILE_PROPERTIES
userlist : P USER
authenticated : USER → P AUTHENTICATION_CONTROL
processlist : P PROCESS

∀p1, p2 : processlist p1.process_id = p2.process_id ⇔ p1 = p2
∀p : processlist p.filename ∈ dom filelist ∧ p.owner ∈ userlist
dom authenticated ⊆ userlist
```

Figure 6. Revised System Specification to Include Authentication

There are three new operations for the authenticated system. The first, consists of two parts: Authenticate_User and Authenticate_User_Fail. Authenticate_User, shown in Figure 7, provides the authentication tactic. It verifies that a user is allowed to access the
system by making sure the username, authentication_control pair is in the authenticated function. The second, Authenticate_User_Fail, also shown in Figure 7, returns the appropriate error message if the username, authentication_control pair does not exist.

\[\text{Authenticate\_User}\]
\[
\begin{array}{l}
\exists \text{SYSTEM} \\
u? : \text{USER} \\
ac? : \text{AUTHENTICATION\_CONTROL} \\
r! : \text{Response} \\
\end{array}
\]

\[ac? \in \text{authenticated}(u?)\]
\[r! = \text{OK}\]

\[\text{Authenticate\_User\_Fail}\]
\[
\begin{array}{l}
\exists \text{SYSTEM} \\
u? : \text{USER} \\
ac? : \text{AUTHENTICATION\_CONTROL} \\
r! : \text{Response} \\
\end{array}
\]

\[ac? \in \text{authenticated}(u?)\]
\[r! = \text{InvalidUserAuthenticationControlPair}\]

Figure 7. Authenticate User Specification

The other two new operations added are AddAuthentication and DeleteAuthentication. They control how the authentication set is modified. Figure 8 shows AddAuthentication. AddAuthentication_OK confirms the user exists before adding the user, authentication_control pair to the authenticated set. If the user doesn’t exist, AddAuthentication_NotExists returns an error. AddAuthentication checks the user
adding the authentication pair is authenticated, then attempts the add.

DeleteAuthentication is similar.

\[ \text{AddAuthentication}_\_\text{OK} \]
\[ \Delta \text{SYSTEM} \]
\[ \text{user}_? : \text{USER} \]
\[ \text{ac}? : \text{AUTHENTICATION\_CONTROL} \]
\[ r! : \text{Response} \]
\[ \text{user}_? \in \text{userlist} \]
\[ \text{authenticated}_\!(\text{user}_?) = \text{authenticated}_\!(\text{user}_?) \cup \{\text{ac}?\} \]
\[ r! = \text{OK} \]

\[ \text{AddAuthentication}_\_\text{NotExists} \]
\[ \exists \text{SYSTEM} \]
\[ \text{user}_? : \text{USER} \]
\[ \text{ac}? : \text{AUTHENTICATION\_CONTROL} \]
\[ r! : \text{Response} \]
\[ \text{user}_? \notin \text{userlist} \]
\[ r! = \text{UserNotExists} \]

\[ \text{AddAuthentication} \equiv (\text{Authenticate}_\text{User} \land (\text{AddAuthentication}_\text{OK} \lor \text{AddAuthentication}_\text{NotExists})) \lor \text{Authenticate}_\text{User}_\text{Fail} \]

Figure 8. Add Authentication Specification

AddAuthentication shows how all the operations on the system need to be modified to add authentication, the user performing the task must first be authorized, then the operation is performed or the authorization fails. Figure 9 shows how AddUser and ChangeFile have been modified.
There are several important features that this specification emphasizes. First, the user is authenticated and then the action is performed, this prevents covert channels where different responses from the system can be used to determine properties of the system. Second, the system as specified performs complete mediation. Complete mediation, where every action is authenticated, is one of the secure design principles [3]. Third, the system, while requiring complete mediation, allows for a Kerberos style authentication scheme where the user is provided with a ticket or authentication control. This scheme allows the user a simpler access control, instead of having to provide their main authentication control for every action. Fourth, the authentication scheme is only as good as the authentication control, if the authentication control is easily compromised, then the system will not be secure. This is a feature of all authentication schemes and this specification make the association explicit. As this system model is refined, the designer will have to determine what constitutes a sufficiently strong authentication control for their system. For example, a system for the military would need stronger authentication than a system for a university. Fifth, the ACM must be created in an appropriate manner, if a user is given rights that they should not have, the information within the system could
be compromised. Finally, the system does not provide authorization. If an entity knows a correct user, authentication_control pair then that entity can perform every operation in the system.

3.4 Authorization Tactic Model Specification

This section presents the authorization tactic model specification. The full specification is in Appendix C.

Authorization is the determination of whether an entity is allowed to perform an action or not. The access control matrix, which describes the rights every user has with respect to every object in the system [3], is one way to implement the authorization tactic. For this system, the access control matrix (ACM) consists of four separate sets, one for each type of object in the base system: userlist, filelist and processlist and one set that controls access to the ACM itself.

Each access control set is a mapping from the user performing the operation and the object being operated on to a set of rights. The rights are defined to coincide with all possible actions on the system. Access to the ACM must also be controlled. Access control for the ACM is defined for each of the four sets, the user is either granted access to a set in the ACM or not. This level of granularity was chosen for the specification because additional granularity, such as controlling whether a user can only add or delete
entities from a given set in the ACM makes the specification more verbose without adding meaningfully to the specification. Figure 10 shows the new system and additional types.

\[
\begin{align*}
\text{UserAuthorization} &::= \text{Add User} | \text{Delete User} \\
\text{FileAuthorization} &::= \text{Add File} | \text{Delete File} | \text{Read File} | \text{Change File} \\
\text{ProcessAuthorization} &::= \text{Start Process} | \text{Stop Process} | \text{Change Process} \\
\text{ACMAuthorization} &::= \text{User} | \text{File} | \text{Process} | ACM
\end{align*}
\]

\[\text{SYSTEM}\]
\[
\begin{align*}
\text{filelist} &: \text{FILENAME} \rightarrow \text{FILE_PROPERTIES} \\
\text{userlist} &: \mathbb{P} \text{ USER} \\
\text{processlist} &: \mathbb{P} \text{ PROCESS} \\
\text{userACM} &: \text{USER} \times \text{USER} \rightarrow \mathbb{P} \text{UserAuthorization} \\
\text{fileACM} &: \text{USER} \times \text{FILENAME} \rightarrow \mathbb{P} \text{FileAuthorization} \\
\text{processACM} &: (\text{USER} \times (\text{USER} \times \text{FILENAME})) \rightarrow \mathbb{P} \text{ProcessAuthorization} \\
\text{acmACM} &: \text{USER} \rightarrow \mathbb{P} \text{ACMAuthorization}
\end{align*}
\]

\[
\begin{align*}
\forall p_1, p_2 : \text{processlist} \bullet p_1.\text{process id} = p_2.\text{process id} \leftrightarrow p_1 = p_2 \\
\forall p : \text{processlist} \bullet p.\text{filename} \in \text{dom filelist} \land p.\text{owner} \in \text{userlist} \\
\text{dom (dom userACM)} \subseteq \text{userlist} \\
\text{dom (dom fileACM)} \subseteq \text{userlist} \\
\text{dom (dom processACM)} \subseteq \text{userlist} \\
\text{dom (ran (dom processACM))} \subseteq \text{userlist} \\
\text{ran (ran (dom processACM))} \subseteq \text{dom filelist} \\
\text{dom acmACM} \subseteq \text{userlist}
\end{align*}
\]

Figure 10. Revised System Specification to Include Authorization

The domains of the four parts of the ACM are subsets of the userlist, a user that doesn’t exist does not have rights within the system. However, a user that doesn’t exist can be an
object in the ACM, in fact, users and files that don’t exist must be objects in the ACM, otherwise creation of objects would not be controlled by the ACM.

The system has four new elements, the sets in the ACM and each element needs to be modifiable, therefore there are eight new operations, an add and a delete for each set in the ACM. Access to these operations is authorized through the acmACM set of the ACM. Figure 11 shows AddUserACM_NewUser. AddUserACM_NewUser takes three users, the first user is the user modifying the UserACM, the second user is the user being granted the right and the third user is the object that the right is being granted for. For example, AddUserACM_OK(u1, u2, u3, {Add_User}) would, assuming u1 has the right to modify the UserACM, allow u2 to add u3 to the userlist. The other operations for adding and deleting entries in the userACM, fileACM, processACM and acmACM are similar.
**AddUserACM_OK**

\[ \Delta \text{SYSTEM} \]

\[ \begin{align*}
user? & : \text{USER} \\
newUser? & : \text{USER} \\
addUser? & : \text{USER} \\
rights? & : \mathcal{P} \text{UserAuthorization} \\
r! & : \text{Response} \\
\end{align*} \]

\[ \begin{align*}
\text{User} & \in \text{acmACM} (user?) \\
\text{newUser} & \in \text{userlist} \\
\text{userACM}' & = \text{userACM} \cup \{(newUser?,addUser?) \rightarrow rights?\} \\
r! & = \text{OK} \\
\end{align*} \]

**AddUserACM_IncorrectRights**

\[ \exists \text{SYSTEM} \]

\[ \begin{align*}
user? & : \text{USER} \\
newUser? & : \text{USER} \\
addUser? & : \text{USER} \\
rights? & : \mathcal{P} \text{UserAuthorization} \\
r! & : \text{Response} \\
\end{align*} \]

\[ \begin{align*}
\text{User} & \notin \text{acmACM} (user?) \\
r! & = \text{IncorrectRights} \\
\end{align*} \]

**AddUserACM_NewUserNotExists**

\[ \exists \text{SYSTEM} \]

\[ \begin{align*}
user? & : \text{USER} \\
newUser? & : \text{USER} \\
addUser? & : \text{USER} \\
rights? & : \mathcal{P} \text{UserAuthorization} \\
r! & : \text{Response} \\
\end{align*} \]

\[ \begin{align*}
\text{User} & \in \text{acmACM} (user?) \\
\text{newUser} & \notin \text{userlist} \\
r! & = \text{UserNotExists} \\
\end{align*} \]

\[ \text{AddUserACM} \equiv \text{AddUserACM_OK} \lor \text{AddUserACM_IncorrectRights} \lor \text{AddUserACM_NewUserNotExists} \]

Figure 11. Add User ACM Specification
Most of the revised operations of the base system are modified in a similar way to authentication, except each operation requires a check that the user has the correct rights, not that the user has the correct authentication control. Figure 12 shows the modified AddUser.

\[ AddUser_{\text{Authorization\_OK}} \]
\[ \exists \text{SYSTEM} \]
\[ \text{user}\? : \text{USER} \]
\[ \text{newUser}\? : \text{USER} \]
\[ r! : \text{Response} \]
\[ \text{Add\_User} \in \text{userACM(user}\?,\text{newUser}\?) \]
\[ r! = \text{OK} \]

\[ AddUser_{\text{Authorization\_Fail}} \]
\[ \exists \text{SYSTEM} \]
\[ \text{user}\? : \text{USER} \]
\[ \text{newUser}\? : \text{USER} \]
\[ r! : \text{Response} \]
\[ \text{Add\_User} \in \text{userACM(user}\?,\text{newUser}\?) \]
\[ r! = \text{IncorrectRights} \]

\[ AddUser \equiv (AddUser_{\text{Authorization\_OK}} \land (AddUser_{\text{OK}} \lor AddUser_{\text{Exists}})) \lor AddUser_{\text{Authorization\_Fail}} \]

Figure 12. Revised Add User Specification to Include Authorization

The operations on processes have been modified to include a runAs user. The idea behind this is a user might want to start a process that has fewer rights than they do, for example, root on a Unix system starting the mailer daemon should not run the mailer
daemon as itself. Figure 13 shows the modified StartProcess operation.

StartProcess_Authorization_OK and StartProcess_Authorization_Fail are virtually identical to the corresponding AddUser operations.

```
StartProcess_OK

ASSYSTEM
process? : PROCESS
runAs? : USER
r! : Response

process?.owner = runAs?
process?.filename ∈ dom_filelist
∀p:processlist • p.process_id ≠ process?.process_id
processlist' = processlist ∪ {process?}
r! = OK
```

```
StartProcess_UserNamesInconsistent

ASSYSTEM
process? : PROCESS
runAs? : USER
r! : Response

process?.owner ≠ runAs?
r! = UserNamesInconsistent
```

```
StartProcess ≡ (StartProcess_Authorization_OK ∧ (StartProcess_OK ∨
   StartProcess_FilenotExists ∨ StartProcess_UserNamesInconsistent))
   ∨ StartProcess_Authorization_Fail
```

Figure 13. Revised Start Process Specification to Include Authorization

The authorization of a system operates similarly to the authentication of a system, it wraps the existing system so a user must be authorized before they can try and perform any action. There are several features of authorization that this specification emphasizes.
First, it does not perform authentication, which is a separate tactic. This does mean that if a system only implements authentication, if someone can imitate a user, they would immediately have that users rights. This is like having a Unix system where all the passwords are empty. In practice, authentication and authorization will almost always be used together. Second, like authentication, this specification performs complete mediation; every operation is checked to make sure the user attempting the operation is allowed to perform that action. Finally, the ACM is capable of representing any style of access control: Roll Based Access Control, Discretionary Access Control or Mandatory Access Control by defining constraints on how the ACM is modified.

3.5 Data Confidentiality Tactic Model Specification

This section presents the data confidentiality tactic model specification. The full specification is in Appendix D.

“Confidentiality is the concealment of information or resources” [3]. Therefore, the date confidentiality tactic requires that the data, or information is hidden. If the information is hidden, then a piece of information is needed to retrieve it. The base system is modified to provide data confidentiality by introducing a new set, SECRET and the mapping to the fileproperties is redefined so it is now from a filename and secret to the fileproperties. Figure 14 shows the revised system specification. The additional constraint at the bottom
of the specification eliminates the possibility of mapping a filename to different properties depending on the secret while still allowing multiple secrets for each filename.

[SECRET]

SYSTEM

- filelist : FILENAME × SECRET → FILE_PROPERTIES
- userlist : ∃ USER
- processlist : ∃ PROCESS

∀p1, p2 : processlist • p1.process_id = p2.process_id ↔ p1 = p2
∀p : processlist • p.filename ∈ dom (dom filelist) ∧ p.owner ∈ userlist
∀f : dom (dom filelist) • ∀s1, s2 : (dom filelist) (\{f\}) • filelist(f, s1) = filelist(f, s2)

Figure 14. Revised System Specification to Include Data Confidentiality

There are no new operations for confidentiality and it does not modify the user operations. All the ReadFile and process operations are changed to include the secret being passed in. If the filename, secret pair is not in the mapping, an error is returned. The system does not return different errors depending on if the filename exists or not. This is important because the existence of the file could be used as a covert channel. Figure 15 shows how ReadFile has been modified.
Figure 15. Revised Read File Specification to Include Data Confidentiality

AddFile is modified to check to see if the filename exists and if it does, if the
fileproperties are the same. If the fileproperties are the same, then AddFile is adding an
additional way to access the file through the new secret. Figure 16 shows how AddFile
has been modified.

Figure 16. Revised Add File Specification to Allow Multiple Secrets per Filename
DeleteFile is modified to remove all references to the filename, not just delete the one way to access the file. ChangeFile is modified to change all filename references to the correct revised fileproperties, see Figure 17.

```
_ChangeFile_OK_

\begin{verbatim}
ASYSTEM
filename? : FILENAME
secret? : SECRET
properties? : FILE_PROPERTIES
r! : Response

(filename?,secret?) ∈ dom filelist
(let secretlist == ((dom filelist) \{filename?\}) •
filelist' = filelist ⊕ ((\{filename?\}×secretlist)×\{properties?\}))

r! = OK
\end{verbatim}

Figure 17. Revised Change File Specification to Include Data Confidentiality
```

There are several features the specification for data confidentiality emphasizes. First, cryptography is not required by this definition, anything that maintains the mapping from the filename, secret pair to the fileproperties will provided confidentiality. The key here is that the mapping is the only way to access the fileproperties. Cryptographic protocols are sufficient, but not necessary. Secure storage mechanisms, such as those proposed by the Trusted Computing Group [15] are an example of a system that provides confidentiality without cryptography. Second, the confidentiality provided is dependant on the quality of the secret, if the secret can be easily guessed or derived, then the
strength of the tactic will be weak. Third, there can be multiple secrets to access a single file. This is useful to prevent users from using the common secret to communicate in any way except through the file.

3.6 Data Integrity Tactic Model Specification

This section presents the data integrity tactic model specification. The full specification is in Appendix E.

The more trustworthy the information, the more integrity it has [3]. Thus, for a system to provide data integrity, the system should be cognizant of any modifications to the information. The data must only be modified by authorized entities and any other modifications to the data must be detected. To provide data integrity, the base system specification needs to be extended; there needs to be two new undefined sets: INTEGRITY_CONTROL and INTEGRITY_CHECK. The integrity_control set is used to make sure only authorized users change a file’s properties. The integrity_check is used to determine if a file has been illegally modified. The integrity_check is stored in the file_properties along with the file_information. The file_information and an integrity_control element are mapped to the integrity_check value. The function, integrity_map, needs to be an injective function to prevent multiple file_info and integrity_control pairs mapping to the same integrity_check. This would prevent the
integrity_check from guaranteeing there has been no unauthorized change. See Figure 18 for the integrity_map and refined file_properties specifications.

\[
\text{INTEGRITY\_MAP : FILE\_INFO} \times \text{INTEGRITY\_CONTROL} \\
\quad \Rightarrow \text{INTEGRITY\_CHECK}
\]

\[
\begin{array}{l}
\text{FILE\_PROPERTIES} \\
\text{info : FILE\_INFO} \\
\text{integrity : INTEGRITY\_CHECK} \\
\exists ic : \text{INTEGRITY\_CONTROL} \bullet \text{INTEGRITY\_MAP(info, ic)} = \text{integrity}
\end{array}
\]

Figure 18. Revised File Properties Specification to Include Data Integrity

There are two new operations for the system Check\_Integrity and Check\_Integrity\_Fail, shown in Figure 19. Check\_Integrity checks to make sure the integrity\_control provided in the operation maps the file\_info to the integrity\_check. Check\_Integrity\_Fail returns the error message if the integrity check is unsuccessful. Note that Check\_Integrity has no return value because it will be included in the successful cases. All the operations include Check\_Integrity in the successful case and Integrity\_Check\_Fail will be an additional error case.
The only operations that change are ReadFile, ChangeFile, StartProcess, StopProcess and ChangeProcess. These are the operations that use or modify an existing file and they all change in the same way. The successful access to the file includes the Integrity_Check specification and the overall operation includes Integrity_Check_Fail. This is different to the way authentication and authorization is specified because the integrity is only checked if the file exists whereas authentication and authorization are always performed. The revised ReadFile specification is shown in Figure 20.
AddFile and ChangeFile check the integrity of the incoming file_properties. If these two operations did not perform the integrity check, the system could end up in an inconsistent state.

There are several noteworthy features of the data integrity specification. First, the requirement that the integrity check be injective is very restrictive. However, without that requirement, it is possible to have multiple file_info and integrity_control pairs map to a single integrity_check. This would violate the system's ability to guarantee that only authorized users modify the files. In most systems, this requirement could be relaxed to insist on a high difficulty in finding the multiple mappings. Second, as with data confidentiality and authorization, the quality of the integrity_control directly affects the quality of the integrity provided. The easier it is to obtain the integrity_control value for a file, the easier it is to circumvent the security mechanism. Third, DeleteFile does not
perform an integrity check. This means that any user can delete files. Implementation of
the authorization tactic allows the system to control who deletes files. Finally, the
three process operations confirm the integrity of the file at every step thus if the file is
ever modified, the execution of that file is halted. This does not prevent buffer overflow
attacks as those attacks don't modify the files, just the information in dynamic memory.
To prevent this class of attacks, the system needs to link the file representation in
memory to the file representation on the hard disk.

3.7 Auditing Tactic Model Specification

This section presents the auditing tactic model specification. The full specification is in
Appendix F.

Auditing a computer system is the process of recording sufficient information for each
event so the sequence of events is reproducible [2]. If a system is then compromised, the
audit log can show the source of the compromise and what was done. Auditing every
operation in a computer system is usually infeasible, the performance impact can be too
large. Therefore, it is important that an audit policy exist which determines the
operations to be logged.

Two elements are added to the base system: the audit policy and the log. The audit
policy consists of three parts, the userauditpolicy, the fileauditpolicy and the
processauditpolicy. Each policy determines what actions on the user, file or process portion of the system needs to be audited. Figure 21 shows the extended system specification.

```
SYSTEM
filelist : FILENAME→FILE_PROPERTIES
userlist : P USER
processlist : P PROCESS
userauditpolicy : P(USER × UserAction)
fileauditpolicy : P(Filename × FileAction)
processauditpolicy : P(USER × FILENAME × ProcessAction)
log : seq LogElement
∀p1, p2 : processlist • p1.process_id = p2.process_id ↔ p1 = p2
∀p : processlist • p.filename ∈ dom filelist ∧ p.owner ∈ userlist
```

Figure 21. Revised System Specification to Include Auditing

The second element added to the system model is the log. The log is a sequence of log elements. Each log element includes the result of the operation performed, and either a UserLogElement, a FileLogElement or a ProcessLogElement depending on the operation. Both successful actions and unsuccessful actions need to be logged because it is important that a system report failures such as logins using incorrect usernames or passwords. The log also needs to keep all operations in a single sequence. If separate sequences are kept for the user, file and process operations, then a timestamp or some other coordination mechanism is required so operations on different parts of the system
are reproduced in the correct order. The log element specifications are shown in Figure 22.

\[ 
\begin{align*}
\text{UserLogElement} & \\
\text{user: USER} & \\
\text{action: UserAction} & \\
\text{FileLogElement} & \\
\text{filename: FILENAME} & \\
\text{action: FileAction} & \\
\text{filechange: FILE_CHANGE} & \\
\text{ProcessLogElement} & \\
\text{process id: PROCESS_ID} & \\
\text{filename: FILENAME} & \\
\text{owner: USER} & \\
\text{action: ProcessAction} & \\
\text{processchange: PROCESS_STATE_CHANGE} & \\
\text{LogElement} & \\
\text{u: UserLogElement} & \\
\text{f: FileLogElement} & \\
\text{p: ProcessLogElement} & \\
\text{r: Response} & \\
\#u + \#f + \#p = 1 & 
\end{align*} 
\]

Figure 22. Log Element Specifications

The file and process log elements keep track of changes to files or processes. This is done through two mappings. The first mapping is from the new and old properties to the change. The second mapping is from the old properties and the change to the new
properties. These two mappings allow the changes to be determined and then repeated.

Figure 23 contains the specification for the mappings.

```plaintext
FILE_CHANGE_MAP : FILE_PROPERTIES × FILE_CHANGE
                    → FILE_PROPERTIES

FILE_PROPERTIES_MAP : FILE_PROPERTIES × FILE_PROPERTIES
                      → FILE_CHANGE

PROCESS_STATE_CHANGE_MAP : PROCESS_STATE × PROCESS_STATE_CHANGE
                           → PROCESS_STATE

PROCESS_STATE_MAP : PROCESS_STATE × PROCESS_STATE
                    → PROCESS_STATE_CHANGE
```

Figure 23. Change in File and Process Properties Function Specifications

All operation cases, including the failure message reports, are modified. Each operation checks if the audit policy requires the operation be logged. If so, the log entry is created and appended to the log. The modified operation case specifications require local or temporary variables, and in Z there are two ways to do this. The first, shown in Figure 24, uses the let expression. Unfortunately, the let expression in Z Word Tools can only evaluate to a set which makes the definition awkward as there are sets of 1 element where the element needs to be added to a sequence. The second, shown in Figure 25, lists the local variables as part of the schema declaration. The local variables have neither a ! or ? so they are neither input nor output. The second definition, while not standard is
acceptable as demonstrated in [5] on page 248. The second specification is easier to read and determine the intent of the specification so it is used. Every operation case except StopProcess_ProcessIDNotExists and ChangeProcess_ProcessIDNotExists is modified in virtually the same way.

\[ \text{ReadFile\_OK} \]

\[
\Delta \text{SYSTEM} \\
\text{filename}\?: \text{FILENAME} \\
\text{fileproperties}\!: \text{FILE\_PROPERTIES} \\
\text{r}\!: \text{Response} \\
\]

\[
\text{filename}\? \in \text{dom filelist} \\
\text{fileproperties}\! = \text{filelist(filename\?)} \\
\text{r}\! = \text{OK} \\
\text{Read\_File} \in \text{fileauditpolicy} \\
(\text{let } f = \{ \text{FileLogElement} \mid \text{f.filename} = \text{filename}\? \land \text{f.action} = \text{Read\_File} \} \land \\
(\#f = 1 \land (\text{let } l = \{ \text{LogElement} \mid \text{l.u} = \emptyset \land \text{l.f} = f \land \text{l.p} = \emptyset \land \text{l.r} = \text{r}\! \}) \land \\
(\forall z : l \bullet \text{log} = \text{log}(z) \land \#l = 1)))
\]

Figure 24. Revised Read File Specification to Include Auditing Using Lets
\begin{align*}
\text{ReadFile\_OK} \\
\text{△SYSTEM} \\
\text{filename? : FILENAME} \\
\text{fileproperties! : FILE\_PROPERTIES} \\
\text{filelogelement : FileLogElement} \\
\text{logelement : LogElement} \\
\text{r! : Response} \\
\text{filename? ∈ dom filelist} \\
\text{fileproperties! = filelist(filename?)} \\
\text{r! = OK} \\
(f\text{ilename?, Read\_File) ∈ fileauditpolicy \vee (filelogelement.filename= filename? \wedge} \\
\text{filelogelement.action = Read\_File} \land \text{logelement.r = r!} \land} \\
\text{logelement.u = ∅ \land \text{logelement.f = \{filelogelement\} \land} \\
\text{logelement.p = ∅ \land log' = log'(logelement))}
\end{align*}

\begin{align*}
\text{ReadFile\_NotExists} \\
\text{△SYSTEM} \\
\text{filename? : FILENAME} \\
\text{fileproperties! : FILE\_PROPERTIES} \\
\text{filelogelement : FileLogElement} \\
\text{logelement : LogElement} \\
\text{r! : Response} \\
\text{filename? ∈ dom filelist} \\
\text{r! = FileNotExists} \\
(f\text{ilename?, Read\_File) ∈ fileauditpolicy \vee (filelogelement.filename= filename? \wedge} \\
\text{filelogelement.action = Read\_File} \land \text{logelement.r = r!} \land} \\
\text{logelement.u = ∅ \land \text{logelement.f = \{filelogelement\} \land} \\
\text{logelement.p = ∅ \land log' = log'(logelement))}
\end{align*}

ReadFile \equiv \text{ReadFile\_OK} \lor \text{ReadFile\_NotExists}

Figure 25. Revised Read File Specification to Include Auditing with Local Variables in the Schema Declaration

StopProcess\_ProcessIDNotExists and ChangeProcess\_ProcessIDNotExists are not modified in the same manner as the other operation cases because the process referenced
does not exist so there is no user, file pair to determine if the case should be logged or not. The specification uses the security principle of fail safe defaults and if there is any user, filename pair that requires logging, then the failure is logged. Figure 26 shows the revised StopProcess_ProcessIDNotExists specification.

\[
\text{StopProcess\_ProcessIDNotExists}\\
\exists \text{SYSTEM}\\
\quad \text{process\_id}: \text{PROCESS\_ID}\\
\quad \text{processlogelement}: \text{ProcessLogElement}\\
\quad \text{logelement}: \text{LogElement}\\
\quad r!: \text{Response}\\
\quad \forall p: \text{processlist} \bullet p.\text{process\_id} \neq \text{process\_id}\\
\quad r! = \text{ProcessIDNotExists}\\
\quad (\forall \text{owner}: \text{USER} \bullet (\forall \text{filename}: \text{FILENAME} \bullet\\
\quad \quad (\text{owner, filename, Stop\_Process} \in \text{processauditpolicy}) \lor \text{processlogelement.action} = \text{Stop\_Process} \land \text{logelement.r} = r! \land \text{logelement.u} = \emptyset \land \text{logelement.f} = \emptyset \land \text{logelement.p} = \{\text{processlogelement}\} \land \log' = \log(\text{logelement}))
\]

Figure 26. Revised StopProcess\_ProcessIDNotExists Specification to Include Auditing

There are several important features of this specification. First, there is no way to delete or modify the log. If log elements can be modified or deleted, users can hide their actions. Second, the audit could log every change to a process or file, this could lead to a severe performance impact therefore the audit policy is very important. It is essential to find the correct balance between logging sufficient information to recreate all critical processes and files and keeping the performance of the system satisfactory. Third, the specification does not allow for the audit policy to be modified while the system is
running, it is set up from the start of the system. If the log policy requires customization, it is a straightforward exercise to add the operations. The operations will be similar to those defined for authorization. Fourth, there can be additional granularity added to the audit policy so logging depends on the result of the operation. This is a straightforward refinement of the system, which is not shown here as it only serves to make the specification more difficult to interpret.

3.8 Intrusion Detection Tactic Model Specification

This section presents the intrusion detection tactic model specification. The full specification is in Appendix G.

An intrusion detection system (IDS) analyses the events in a system to determine if they constitute an attack [3]. An intrusion detection system requires two sets of information to make this determination. First it needs a policy to define an attack, second it needs a record of the events occurring within the system. The audit tactic provides a record of the events, so the intrusion detection system specification will extend the audit system specification.

The system is modified by adding three new elements: the IDS Policy, the alertlist and the lastIDSAnalysis number. The IDS policy is a set of log element sequences where each sequence represents an attack. The alertlist is a set of log element sequences where
each sequence represents a detected attack. The lastIDSAnalysis number represents the last log entry analyzed by the IDS. See Figure 27 for the extended system specification.

\[
\text{SYSTEM} \\
\text{filelist} : \text{FILENAME} \rightarrow \text{FILE_PROPERTIES} \\
\text{userlist} : \mathbb{P} \text{ USER} \\
\text{processlist} : \mathbb{P} \text{ PROCESS} \\
\text{userlogpolicy} : \mathbb{P}(\text{USER} \times \text{UserAction}) \\
\text{filelogpolicy} : \mathbb{P}(\text{FILENAME} \times \text{FileAction}) \\
\text{processlogpolicy} : \mathbb{P}(\text{USER} \times \text{FILENAME} \times \text{ProcessAction}) \\
\text{log} : \text{seq LogElement} \\
\text{idspolicy} : \mathbb{P}(\text{seq1 LogElement}) \\
\text{idsalertlist} : \text{seq(seq1 LogElement)} \\
\text{lastIDSAnalysis} : \mathbb{N}
\]

\[
\forall p_1, p_2 : \text{processlist} \bullet p_1.\text{process_id} = p_2.\text{process_id} \leftrightarrow p_1 = p_2 \\
\forall p : \text{processlist} \bullet p.\text{filename} \in \text{dom filelist} \land p.\text{owner} \in \text{userlist}
\]

Figure 27. Revised System Specification to Include Intrusion Detection

There are three new operations in the specification, IDS_Alert, IDS_NoAlert and IDS_System, shown in Figure 28. IDS_NoAlert requires that if the log does not change, then there can be no new attacks detected. IDS_Alert analyzes the entire log and if any sequence can be extracted from the log that meets the following two requirements then an alert will be generated: first, the sequence is in the IDSPolicy and second, it contains at least one log element that has not been previously analyzed. This second requirement is to make sure no duplicate alerts are generated. IDS_System is either IDS_Alert or IDS_NoAlert. If IDS_Alert is the only option included, then every operation fails if log'
= \log. The combined operation is therefore required because an operation should not fail because the operation is not logged.

\[ IDS_{\text{Alert}} \]

\[ \Delta \text{SYSTEM} \]

\[
\text{sublog} : \mathbb{P}(\mathbb{P}^0) \\
\text{newalertlog} : \text{seq}(\text{seq}\ \text{LogElement})
\]

\[
\log' \neq \log \\
\forall s : \text{sublog} \bullet s \mid \log' \in \text{idspolicy} \land (s \mid \log') \in \text{newalertlog} \land \\
(\exists p : \mathbb{N}_i \bullet (\text{lastIDSAnalysis} < p \leq \#\log' \land p \in s))
\]

\[
\text{idsalertlist}' = \text{idsalertlist}' \text{newalertlog}
\]

\[
\text{lastIDSAnalysis}' = \#\log'
\]

\[ IDS_{\text{NoAlert}} \]

\[ \Xi \text{SYSTEM} \]

\[
\log' = \log \\
\text{idsalertlist}' = \text{idsalertlist}
\]

\[ IDS_{\text{System}} \equiv IDS_{\text{Alert}} \lor IDS_{\text{NoAlert}} \]

Figure 28. IDS_Alar, IDS_NoAlert and IDS_System Specification

Every operation is modified to include the IDS_System. If the operation is logged, then IDS_Alar will determine if any new attacks have occurred. See Figure 29 for the revised AddUser specification. All other operations are similarly modified.
There are several important features of this specification. First, IDS_Alert is called every
time the log is modified to provide complete mediation. Second, the intrusion detection
system only monitors logged operations, if an attack contains operations that are not
logged, that attack will not be detected, thus the log policy is a crucial part of the intrusion detection system. Third, the intrusion detection policy is predetermined. The system can be extended to allow changes in the policy by following the same pattern as the authorization system. Finally, the intrusion detection policy contains a list of all attacks. This corresponds to the definition of a misuse based IDS. There are two other types of IDS, specification based and anomaly based IDS. These two IDS specify allowable behavior, the opposite of a misuse based system. In theory, the conversion between the two types of IDS is not difficult, however, in a real system specifying all allowable behavior or all illegal behavior is infeasible.

3.9 Limit Access, Limit Exposure and Availability Tactics

Availability is a quality attribute like security [2]. As such, there is a set of tactics that can be used to improve the availability of a system. This report is focused on security tactics and does not investigate availability tactics.

The limit exposure and limit access tactics do not require separate specifications because they can be achieved through a combination of the other tactics. The idea behind the limit exposure tactic is to reduce the number of services on a given host so that there are a limited number of ways to get into the system. This tactic has two possible interpretations. The first is that the system being defined is the system containing multiple services. Under the limit exposure tactic, the services need to be kept separate.
This is sandboxing, separating the services therefore they cannot interact. This tactic can be achieved by using authentication and authorization and constructing the access control matrix of the system so that it partitions the files, processes, and users so the services cannot interact. It can be further strengthened by using confidentiality and integrity where each service has its own secrets and integrity_controls unbeknownst to the other services, thus if one service is compromised, it cannot access the other services and if it indeed accessed them, it would not be able to modify or access the other services files.

The second interpretation of the limit exposure tactic is that the system being defined is within another system. In this case access control would not be helpful, however confidentiality and integrity would prevent other services from reading or manipulating the files of this service. The best way to get this separation is to have the system sandboxed, however, if that is not possible, then using the confidentiality and integrity tactics can provide some protection.

The idea behind the limit access tactic is to limit the locations or access points able to connect to the system. An example of a mechanism that implements the limit access tactic is a firewall. A firewall is outside of the system being defined, so this specification does not define the properties of the firewall. However, the definition of authorization_control can be extended to include the location of the user and proof of that location. Authorization can then be used to limit system access to users in permitted locations.
Both of these tactics are primarily meant for the deployment stage of the system being defined, therefore in general it is not something that the software architecture will deal with. However, if they are important, there are mechanisms that can be implemented in the software system to make these tactics stronger.

3.10 Combined Authorization and Authentication Model Specification

This section presents the combined authorization and authentication tactic model specification. The full specification is in Appendix H.

Most software systems employ more than one security quality attribute thus most software architectures require more than one security tactic. The formal specifications facilitate combining. As a representative example, the merged authentication and authorization specification will be developed. In this specification, the order of operations is important; authorization needs to be performed before authentication otherwise it is possible for an unauthorized entity to map out the rights of the authorized users. The system is the combination of the authentication system and the authorization system. Furthermore, access to this new authorization set is controlled by the access control matrix. Figure 30 shows the combined authorization and authentication system specification.
The new portion of the ACM, the authentication ACM, needs to be modified similar to all the other ACM components. The two operations, AddAuthenticationACM and DeleteAuthenticationACM are virtually identical and show how authorization and authentication are combined. First, the user is authenticated and then the user’s right to modify the ACM is corroborated. See Figure 31.
**AddAuthenticationACM_OK**

\[\begin{align*}
\Delta & \text{SYSTEM} \\
\text{addUser} & : \text{USER} \\
\text{newUser} & : \text{USER} \\
\text{user} & : \text{USER} \\
\text{rights} & : \mathbb{P}_{\text{AuthenticationAuthorization}} \\
\text{r} & : \text{Response}
\end{align*}\]

\[\begin{align*}
\text{Authentication} & \in \text{acmACM}(\text{user}) \\
\text{newUser} & \in \text{userlist} \\
\text{authenticationACM}' & = \text{authenticationACM} \cup \{(\text{newUser}, \text{addUser})\rightarrow \text{rights}\} \\
\text{r} & = \text{OK}
\end{align*}\]

**AddAuthenticationACM_IncorrectRights**

\[\begin{align*}
\exists & \text{SYSTEM} \\
\text{addUser} & : \text{USER} \\
\text{newUser} & : \text{USER} \\
\text{user} & : \text{USER} \\
\text{rights} & : \mathbb{P}_{\text{AuthenticationAuthorization}} \\
\text{r} & : \text{Response}
\end{align*}\]

\[\begin{align*}
\text{Authentication} & \in \text{acmACM}(\text{user}) \\
\text{r} & = \text{IncorrectRights}
\end{align*}\]

**AddAuthenticationACM_NewUserNotExists**

\[\begin{align*}
\exists & \text{SYSTEM} \\
\text{addUser} & : \text{USER} \\
\text{newUser} & : \text{USER} \\
\text{user} & : \text{USER} \\
\text{rights} & : \mathbb{P}_{\text{AuthenticationAuthorization}} \\
\text{r} & : \text{Response}
\end{align*}\]

\[\begin{align*}
\text{Authentication} & \in \text{acmACM}(\text{user}) \\
\text{newUser} & \notin \text{userlist} \\
\text{r} & = \text{UserNotExists}
\end{align*}\]

\[\text{AddAuthenticationACM} \equiv (\text{Authenticate User} \land (\text{AddAuthenticationACM_OK} \lor \text{AddAuthenticationACM_IncorrectRights} \lor \text{AddAuthenticationACM_NewUserNotExists})) \lor \text{Authenticate User Fail}\]

Figure 31. AddAuthenticationACM Specification
All operations are defined in the same manner as AddAuthenticationACM, where first the user is authenticated then authorized. The complete specification is a matter of constraining the new system with both tactics. The system now authorizes and authenticates every operation, it performs complete mediation using both security tactics.
Chapter 4

DISCUSSION

4.1 Conclusion

The correct architecture for a system is essential for its successful implementation. Creation of the software architecture is the first and most important stage for realization of the qualities that a system must possess. Security is one of the quality attributes of a system. This project provides Z specifications of architectural tactics that can be used to achieve the security goals of a system. A base system model, with no security is specified and each tactic is defined as constraints on the extended base system model. An example is created which shows how the tactics can be combined into a single specification. This project demonstrates the power of creating Z specifications of architectural tactics and formal languages capacity to capture high-level abstract concepts.

4.2 Applications and Uses

Formal specifications of software systems provide myriads of benefits [12]. The formal specification of the design tactics for the security quality attribute is no exception. First, it provides a template or building block for creating systems that make use of the various tactics. For example, if a system requires confidentiality, then the specification described herein can be used as a base or guide for the architecture of the new system. The generic
nature of the tactic specifications allows them to be integrated into the specification of any type of software system. The ease of adding a specific tactic to a new system allows the software architect to concentrate on the overall design rather than being concerned with the specifics of each tactic.

The formal specifications of the tactics clarify the requirements of each tactic. A tactic is a design decision and requires certain mechanisms or components to be effective. For example, for an intrusion detection system to be effective, there needs to be both a logging policy and an intrusion detection policy both of which capture the security critical behavior of the system. The tactics also require specifications of the quality of the mechanisms. For confidentiality, what the secret is affects how confidential the data is. When the system specification is refined the secret will need to be precisely defined, will it be a password, a digital certificate or something else. A password that acts as the key for a cryptographic algorithm may be sufficient for the system being designed, or a more complex secret such as information on a smartcard may be required. The specification makes explicit the decisions the architect will make regarding the security of the system.

A formally defined system can also be rigorously analyzed. The analysis for potential weaknesses is strengthened with the security aspects of a system precisely defined. As the system specification is refined, the refinements are proven to be consistent with the higher level specifications providing a high level of confidence that the system correctly
implements the tactic. The refinement also exposes the assumptions made about the system. Analysis of the assumptions can then determine if they are acceptable for the system being developed.

The tactic specifications create a framework in which to analyze specific security mechanisms. The mechanism can be proven to correctly capture one or more of the tactics. For example, the Role Based Access Control models specified in [1] can be shown to correctly implement the authorization and authentication tactics.

4.3 Future Work

There are several avenues of additional research that have become apparent during this project. First, it is necessary to formally specify the other quality attributes. As the tactics are formally defined, all that is necessary to create a formal model of a new architecture is to decide what tactics the new system should implement. Further, the formal specification of other quality attributes tactics allows a rigorous analysis of the interactions and compromises that come from combining the tactics.

Second, the tactics can be refined to create a framework or skeleton of a system using programs like those defined in [9], [10] and [11]. This framework will allow the architecture of a system to be implemented quickly and easily. It will drastically shrink
the semantic gap between the architecture, high level design and implementation of the system.

Finally, as many of the tactics require policies such as an intrusion detection or logging policy, there is an area of additional work to create templates for the aforementioned policies should be created. These policies would define various levels of security, from the highest or most strict policies (e.g. for military and other high security applications) to more relaxed policies suited for home or typical business applications. These templates will help to reduce the errors in system development that are caused because of inadequate analysis or definition.
APPENDIX A

Base System Model Specification

[USER]
[FILENAME]
[FILE_PROPERTIES]
[PROCESS_ID]
[PROCESS_STATE]
Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|
ProcessIDNotExists|UserRunningProcess|FileBeingExecuted

PROCESS
filename : FILENAME
owner : USER
process_id : PROCESS_ID
state : PROCESS_STATE

SYSTEM
filelist : FILENAME→FILE_PROPERTIES
userlist : P USER
processlist : P PROCESS
∀p1, p2 : processlist • p1.process_id = p2.process_id ⇒ p1 = p2
∀p : processlist • p.filename ∈ dom filelist ∧ p.owner ∈ userlist

_InitSYSTEM
system : SYSTEM
#system.filelist = 1
#system.userlist = 1
#system.processlist = 1
AddUser_OK
∆SYSTEM
newUser? : USER
r! : Response

newUser? ∈ userlist
userlist' = userlist ∪ {newUser?}
r! = OK

AddUser_Exists
∃SYSTEM
newUser? : USER
r! : Response

newUser? ∈ userlist
r! = UserExists

AddUser = AddUser_OK ∨ AddUser_Exists

DeleteUser_OK
∆SYSTEM
user? : USER
r! : Response

user? ∈ userlist
∀p : processlist • p.owner ≠ user?
userlist' = userlist \ {user?}
r! = OK
DeleteUser_NotExists

\[\exists \text{SYSTEM} \]
user? : USER
r! : Response

user? \notin \text{userlist}
r! = UserNotExists

DeleteUser_RunningProcess

\[\exists \text{SYSTEM} \]
user? : USER
r! : Response

user? \in \text{userlist}
\exists p : \text{processlist} \bullet p.\text{owner} = user?
r! = UserRunningProcess

DeleteUse \equiv \text{DeleteUser_OK} \lor \text{DeleteUser_NotExists} \lor \text{DeleteUser_RunningProcess}

AddFile_OK

\[\Delta \text{SYSTEM} \]
filename? : FILENAME
fileproperties? : FILE_PROPERTIES
r! : Response

filename? \in \text{dom filelist}
filelist' = filelist \cup \{filename? \rightarrow \text{fileproperties}?\}
r! = OK
\textit{AddFile\_Exists} \\
\underline{\exists SYSTEM} \\
\quad \text{filename}? : FILENAME \\
\quad \text{fileproperties}? : FILE\_PROPERTIES \\
\quad r! : \text{Response} \\
\underline{\text{filename}? \in \text{dom filelist}} \\
\underline{r! = \text{FileExists}} \\

AddFile \equiv \text{AddFile\_OK} \lor \text{AddFile\_Exists} \\

\textit{ReadFile\_OK} \\
\underline{\exists SYSTEM} \\
\quad \text{filename}? : FILENAME \\
\quad \text{fileproperties}!: \text{FILE\_PROPERTIES} \\
\quad r! : \text{Response} \\
\underline{\text{filename}? \in \text{dom filelist}} \\
\underline{\text{fileproperties}! = \text{filelist}(\text{filename}?)} \\
\underline{r! = \text{OK}} \\

\textit{ReadFile\_NotExists} \\
\underline{\exists SYSTEM} \\
\quad \text{filename}? : FILENAME \\
\quad \text{fileproperties}!: \text{FILE\_PROPERTIES} \\
\quad r! : \text{Response} \\
\underline{\text{filename}? \notin \text{dom filelist}} \\
\underline{r! = \text{FileNotFoundException}} \\

ReadFile \equiv \text{ReadFile\_OK} \lor \text{ReadFile\_NotExists}
DeleteFile_OK

\[\Delta \text{SYSTEM}\]
filename? : FILENAME
r! : Response

\[\text{filename}? \in \text{dom filelist}\]
\[\forall p : \text{processlist} \bullet p.\text{filename} \neq \text{filename}?\]
\[\text{filelist'} = \text{filelist} \setminus \{\text{filename}? \mapsto \text{filelist}(\text{filename}?)\}\]
\[r! = \text{OK}\]

DeleteFile_NotExists

\[\exists \text{SYSTEM}\]
filename? : FILENAME
r! : Response

\[\text{filename}? \in \text{dom filelist}\]
\[r! = \text{FileNotExists}\]

DeleteFile_FileBeingExecuted

\[\exists \text{SYSTEM}\]
filename? : FILENAME
r! : Response

\[\text{filename}? \in \text{dom filelist}\]
\[\exists p : \text{processlist} \bullet p.\text{filename} = \text{filename}?\]
\[r! = \text{FileBeingExecuted}\]

DeleteFile \equiv \text{DeleteFile_OK} \lor \text{DeleteFile_NotExists} \lor
\text{DeleteFile_FileBeingExecuted}
ChangeFile_OK

\[\Delta \text{SYSTEM}

\text{filename}?: \text{FILENAME} \\
\text{properties}?: \text{FILE\_PROPERTIES} \\
r!: \text{Response}

\text{filename}? \in \text{dom filelist} \\
\text{filelist}' = \text{filelist} \oplus \{\text{filename}? \mapsto \text{properties}??\} \\
r! = \text{OK}

\]

ChangeFile_NotExists

\[\exists \text{SYSTEM}

\text{filename}?: \text{FILENAME} \\
\text{properties}?: \text{FILE\_PROPERTIES} \\
r!: \text{Response}

\text{filename}? \in \text{dom filelist} \\
r! = \text{FileNotFoundException}

\]

\(\text{ChangeFile} \equiv \text{ChangeFile\_OK} \lor \text{ChangeFile\_NotExists}\)

StartProcess_OK

\[\Delta \text{SYSTEM}

\text{process}?: \text{PROCESS} \\
r!: \text{Response}

\text{process}?.\text{owner} \in \text{userlist} \\
\text{process}?.\text{filename} \in \text{dom filelist} \\
\forall p:\text{processlist} \bullet p.\text{process\_id} \neq \text{process}?.\text{process\_id} \\
\text{processlist}' = \text{processlist} \cup \{\text{process}?\} \\
r! = \text{OK}

\]
\texttt{StartProcess\_FileNotExists}

\hspace{1em} \triangleright \texttt{SYSTEM}
\hspace{1em} \texttt{process? : PROCESS}
\hspace{1em} \texttt{r! : Response}

\hspace{2em} \texttt{process?.filename} \notin \texttt{dom\ filelist}
\hspace{1em} \texttt{r! = FileNotExists}

\texttt{StartProcess\_UserNotExists}

\hspace{1em} \triangleright \texttt{SYSTEM}
\hspace{1em} \texttt{process? : PROCESS}
\hspace{1em} \texttt{r! : Response}

\hspace{2em} \texttt{process?.owner} \notin \texttt{userlist}
\hspace{1em} \texttt{r! = UserNotExists}

\texttt{StartProcess} \equiv \texttt{StartProcess\_OK} \lor \texttt{StartProcess\_FileNotExists} \lor
\hspace{1em} \texttt{StartProcess\_UserNotExists}

\texttt{StopProcess\_OK}

\hspace{1em} \triangleright \texttt{SYSTEM}
\hspace{1em} \texttt{process\_id? : PROCESS\_ID}
\hspace{1em} \texttt{r! : Response}

\hspace{2em} \exists p:\texttt{processlist} \bullet p\.process\_id = process\_id?
\hspace{1em} \texttt{processlist'} = \texttt{processlist}\{p:PROCESS|p\.process\_id = process\_id?\}
\hspace{1em} \texttt{r! = OK}

\texttt{StopProcess\_ProcessIDNotExists}

\hspace{1em} \triangleright \texttt{SYSTEM}
\hspace{1em} \texttt{process\_id? : PROCESS\_ID}
\hspace{1em} \texttt{r! : Response}

\hspace{2em} \forall p:\texttt{processlist} \bullet p\.process\_id \neq process\_id?
\hspace{1em} \texttt{r! = ProcessIDNotExists}
StopProcess ≡ StopProcess_OK ∨ StopProcess_ProcessIDNotExists

\[ \Delta \text{SYSTEM} \]
\[
\text{process\_id}\ ? : \text{PROCESS\_ID} \\
\text{newstate}\ ? : \text{PROCESS\_STATE} \\
\text{r!} : \text{Response} \\
\]

\[(\text{let } p \equiv \{ p1 : \text{processlist} \mid p1.\text{process\_id} = \text{process\_id}\?\} \bullet \\
(\forall p2 : p \bullet p2.\text{state} = \text{newstate}\?) \land \# p = 1 \land \\
\text{processlist}' = \text{processlist}\setminus \{ p2 : \text{processlist} \mid p2.\text{process\_id} = \text{process\_id}\?\} \land \\
\text{processlist}' = \text{processlist}' \cup p \land \\
\text{r!} = \text{OK} )\]

\[ \Xi \text{SYSTEM} \]
\[
\text{process\_id}\ ? : \text{PROCESS\_ID} \\
\text{newstate}\ ? : \text{PROCESS\_STATE} \\
\text{r!} : \text{Response} \\
\]

\[(\forall p : \text{processlist} \bullet p.\text{process\_id} \neq \text{process\_id}\? \\
\text{r!} = \text{ProcessIDNotExists} )\]

\[ \text{ChangeProcess} ≡ \text{ChangeProcess\_OK} ∨ \text{ChangeProcess\_ProcessIDNotExists} \]
APPENDIX B

Authentication Tactic System Model Specification

\[
\begin{align*}
&\text{[USER]} \\
&\text{[FILENAME]} \\
&\text{[FILE_PROPERTIES]} \\
&\text{[PROCESS_ID]} \\
&\text{[PROCESS_STATE]} \\
&\text{[AUTHENTICATION\_CONTROL]} \\
\text{Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|} \\
&\quad|ProcessIDNotExists|InvalidUserAuthenticationControlPair| \\
&\quad|UserRunningProcess|FileBeingExecuted|UserStillAuthenticated
\end{align*}
\]

\[
\begin{align*}
&\text{PROCESS} \\
&\quad\text{filename: FILENAME} \\
&\quad\text{owner: USER} \\
&\quad\text{process_id: PROCESS\_ID} \\
&\quad\text{state: PROCESS\_STATE}
\end{align*}
\]

\[
\begin{align*}
&\text{SYSTEM} \\
&\quad\text{filelist: FILENAME\rightarrow FILE\_PROPERTIES} \\
&\quad\text{userlist: \mathbb{P}\\text{USER}} \\
&\quad\text{authenticated: USER\rightarrow \mathbb{P}\text{AUTHENTICATION\_CONTROL}} \\
&\quad\text{processlist: \mathbb{P}\\text{PROCESS}} \\
&\quad\forall p_1, p_2: \text{processlist} \bullet p_1.\text{process_id} = p_2.\text{process_id} \Leftrightarrow p_1 = p_2 \\
&\quad\forall p: \text{processlist} \bullet p.\text{filename} \in \text{dom filelist} \land p.\text{owner} \in \text{userlist} \\
&\text{dom authenticated} \subseteq \text{userlist}
\end{align*}
\]
Authenticate_User

\[\text{Authenticate\_User}\]

\[\exists \text{SYSTEM}\]
\[u\,? : \text{USER}\]
\[ac\,? : \text{AUTHENTICATION\_CONTROL}\]
\[r! : \text{Response}\]

\[ac\,? \in \text{authenticated}(u\,?)\]
\[r! = \text{OK}\]

Authenticate_User_Fail

\[\text{Authenticate\_User\_Fail}\]

\[\exists \text{SYSTEM}\]
\[u\,? : \text{USER}\]
\[ac\,? : \text{AUTHENTICATION\_CONTROL}\]
\[r! : \text{Response}\]

\[ac\,? \notin \text{authenticated}(u\,?)\]
\[r! = \text{InvalidUserAuthenticationControlPair}\]

AddAuthentication_OK

\[\Delta \text{SYSTEM}\]
\[user\,? : \text{USER}\]
\[ac\,? : \text{AUTHENTICATION\_CONTROL}\]
\[r! : \text{Response}\]

\[\text{user}\,? \in \text{userlist}\]
\[\text{authenticated}'(\text{user}\,?) = \text{authenticated}(\text{user}\,?) \cup \{ac\,?\}\]
\[r! = \text{OK}\]

AddAuthentication_NotExists

\[\exists \text{SYSTEM}\]
\[user\,? : \text{USER}\]
\[ac\,? : \text{AUTHENTICATION\_CONTROL}\]
\[r! : \text{Response}\]

\[\text{user}\,? \notin \text{userlist}\]
\[r! = \text{UserNotExists}\]
AddAuthentication ≡ (Authenticate_User ∧ (AddAuthentication_OK ∨ AddAuthentication_NotExists)) ∨ Authenticate_User_Fail

_DeleteAuthentication_OK__

\[ \Delta \text{SYSTEM} \]
user? : USER
ac? : AUTHENTICATION_CONTROL
r! : Response

\[ ac? \in \text{authenticated}(user?) \]
\[ \text{authenticated}'(user?) = \text{authenticated}(user?) \setminus \{ac?\} \]

_DeleteAuthentication_NotExists__

\[ \exists \text{SYSTEM} \]
user? : USER
ac? : AUTHENTICATION_CONTROL
r! : Response

\[ ac? \notin \text{authenticated}(user?) \]
\[ r! = \text{InvalidUserAuthenticationControlPair} \]

DeleteAuthentication ≡ (Authenticate_User ∧ (DeleteAuthentication_OK ∨ DeleteAuthentication_NotExists)) ∨ Authenticate_User_Fail

_InitSYSTEM__

system : SYSTEM

\[ \#\text{system.filelist} = 1 \]
\[ \#\text{system.userlist} = 1 \]
\[ \#\text{system.processlist} = 1 \]
\[ \exists u : \text{system.userlist} \bullet \#(\text{system.authenticated}(u)) \geq 1 \]
AddUser_OK

\[ \text{\Delta \text{SYSTEM}} \]
\[ \text{newUser? : USER} \]
\[ r! : \text{Response} \]
\[ \text{newUser?} \in \text{userlist} \]
\[ \text{userlist'} = \text{userlist} \cup \{\text{newUser?}\} \]
\[ r! = \text{OK} \]

AddUser_Exists

\[ \equiv \text{SYSTEM} \]
\[ \text{newUser? : USER} \]
\[ r! : \text{Response} \]
\[ \text{newUser?} \in \text{userlist} \]
\[ r! = \text{UserExists} \]

AddUser \equiv (\text{Authenticate User} \land (\text{AddUser_OK} \lor \text{AddUser_Exists})) \lor \text{Authenticate User_Fail}

DeleteUser_OK

\[ \text{\Delta \text{SYSTEM}} \]
\[ \text{user? : USER} \]
\[ r! : \text{Response} \]
\[ \text{user?} \in \text{userlist} \]
\[ \forall p : \text{processlist} \cdot p.\text{owner} \neq \text{user}? \]
\[ \text{user?} \in \text{dom authenticated} \]
\[ \text{userlist'} = \text{userlist} \setminus \{\text{user?}\} \]
\[ r! = \text{OK} \]
\[
\_\text{DeleteUser\_NotExists}\]
\begin{align*}
\exists \text{SYSTEM} \\
\text{user}? & : \text{USER} \\
\text{r!} & : \text{Response} \\
\text{user}? & \notin \text{userlist} \\
\text{r!} & = \text{UserNotExists}
\end{align*}

\[
\_\text{DeleteUser\_RunningProcess}\]
\begin{align*}
\exists \text{SYSTEM} \\
\text{user}? & : \text{USER} \\
\text{r!} & : \text{Response} \\
\text{user}? & \in \text{userlist} \\
\exists p : \text{processlist} \cdot p.\text{owner} & = \text{user}? \\
\text{r!} & = \text{UserRunningProcess}
\end{align*}

\[
\_\text{DeleteUser\_StillAuthenticated}\]
\begin{align*}
\exists \text{SYSTEM} \\
\text{user}? & : \text{USER} \\
\text{r!} & : \text{Response} \\
\text{user}? & \in \text{userlist} \\
\forall p : \text{processlist} \cdot p.\text{owner} & \neq \text{user}? \\
\text{user}? & \in \text{dom authenticated} \\
\text{r!} & = \text{UserStillAuthenticated}
\end{align*}

\[
\text{DeleteUser} \equiv (\text{Authenticate\_User} \land (\text{DeleteUser\_OK} \lor \text{DeleteUser\_NotExists} \lor \text{DeleteUser\_RunningProcess} \lor \text{DeleteUser\_StillAuthenticated}) \lor \text{Authenticate\_User\_Fail})
\]
AddFile_OK

\[ \triangledown \text{SYSTEM} \]
\[
\begin{align*}
\text{filename?} &: \text{FILENAME} \\
\text{fileproperties?} &: \text{FILE_PROPERTIES} \\
\text{r!} &: \text{Response} \\
\end{align*}
\]

\[ \text{filename?} \in \text{dom filelist} \]
\[ \text{filelist}' = \text{filelist} \cup \{\text{filename?} \mapsto \text{fileproperties?}\} \]
\[ \text{r!} = \text{OK} \]

AddFile_Exists

\[ \exists \text{SYSTEM} \]
\[
\begin{align*}
\text{filename?} &: \text{FILENAME} \\
\text{fileproperties?} &: \text{FILE_PROPERTIES} \\
\text{r!} &: \text{Response} \\
\end{align*}
\]

\[ \text{filename?} \in \text{dom filelist} \]
\[ \text{r!} = \text{FileExists} \]

AddFile = (Authenticate_User \land (AddFile_OK \lor AddFile_Exists)) \lor Authenticate_User_Fail

ReadFile_OK

\[ \exists \text{SYSTEM} \]
\[
\begin{align*}
\text{filename?} &: \text{FILENAME} \\
\text{fileproperties!} &: \text{FILE_PROPERTIES} \\
\text{r!} &: \text{Response} \\
\end{align*}
\]

\[ \text{filename?} \in \text{dom filelist} \]
\[ \text{fileproperties!} = \text{filelist(filename?)} \]
\[ \text{r!} = \text{OK} \]
\[ \text{ReadFile\_NotExists} \]

\[ \exists \text{SYSTEM} \]

\[ \text{filename}? : \text{FILENAME} \]

\[ \text{fileproperties!} : \text{FILE\_PROPERTIES} \]

\[ \text{r!} : \text{Response} \]

\[ \text{filename}? \in \text{dom filelist} \]

\[ \text{r!} = \text{FileNotExists} \]

\[ \text{ReadFile} \equiv (\text{Authenticate\_User} \land (\text{ReadFile\_OK} \lor \text{ReadFile\_NotExists})) \lor \text{Authenticate\_User\_Fail} \]

\[ \text{DeleteFile\_OK} \]

\[ \Delta \text{SYSTEM} \]

\[ \text{filename}? : \text{FILENAME} \]

\[ \text{r!} : \text{Response} \]

\[ \text{filename}? \in \text{dom filelist} \]

\[ \forall p : \text{processlist} \land p\_filename \neq \text{filename}? \]

\[ \text{filelist}' = \text{filelist} \setminus \{\text{filename}?\mapsto \text{filelist}(\text{filename}?)\} \]

\[ \text{r!} = \text{OK} \]

\[ \text{DeleteFile\_NotExists} \]

\[ \exists \text{SYSTEM} \]

\[ \text{filename}? : \text{FILENAME} \]

\[ \text{r!} : \text{Response} \]

\[ \text{filename}? \notin \text{dom filelist} \]

\[ \text{r!} = \text{FileNotExists} \]
_DeleteFile_FileBeingExecuted_

$$\exists SYSTEM$$  
filename? : FILENAME  
r! : Response  

filename? \in \text{dom filelist}  
$$\exists p : \text{processlist} \bullet p.\text{filename} = \text{filename}?$$  
r! = \text{FileBeingExecuted}


$$\text{DeleteFile} \equiv (\text{Authenticate User} \land (\text{DeleteFile_OK} \lor \text{DeleteFile_NotExists} \lor$$  
$$\text{DeleteFile_FileBeingExecuted})) \lor \text{Authenticate User_Fail}$$


_ChangeFile_OK_

$$\Delta SYSTEM$$  
filename? : FILENAME  
properties? : FILE_PROPERTIES  
r! : Response  

filename? \in \text{dom filelist}  
filelist' = \text{filelist} \oplus \{\text{filename}? \mapsto \text{properties}?\}  
r! = \text{OK}


_ChangeFile_NotExists_

$$\exists SYSTEM$$  
filename? : FILENAME  
properties? : FILE_PROPERTIES  
r! : Response  

filename? \in \text{dom filelist}  
r! = \text{FileNotExists}


$$\text{ChangeFile} \equiv (\text{Authenticate User} \land (\text{ChangeFile_OK} \lor \text{ChangeFile_NotExists}))$$  
$$\lor \text{Authenticate User_Fail}$$
StartProcess_OK

\( \triangle SYSTEM \)
process_id?: PROCESS_ID
newstate?: PROCESS_STATE
\( r! : Response \)

\( (let p = \{ p1 : processlist | p1.process_id = process_id? \} \bullet \)
(\( \bigvee p2 : p \bullet p2.state = newstate? \) \& \#p = 1 \&
processlist' = processlist \{ p2 : processlist | p2.process_id = process_id? \} \&
processlist' = processlist' \cup p \&
\( r! = OK \) )

StartProcess_FileNotExists

\( \exists SYSTEM \)
process?: PROCESS
\( r! : Response \)

process?.filename \( \notin \) dom filelist
\( r! = FileNotExists \)

StartProcess_UserNotExists

\( \exists SYSTEM \)
process?: PROCESS
\( r! : Response \)

process?.owner \( \notin \) userlist
\( r! = UserNotExists \)

StartProcess \( \equiv \) (Authenticate_User \& (StartProcess_OK \lor
StartProcess_FileNotExists \lor StartProcess_UserNotExists))
\lor Authenticate_User_Fail
_StopProcess_OK_

\[ \text{\textsc{\small SYSTEM}} \]
\[ \text{process\_id\?: PROCESS\_ID} \]
\[ r! : \text{Response} \]
\[ \exists p : \text{processlist \bullet p.process\_id = process\_id?} \]
\[ \text{processlist'} = \text{processlist\{p:PROCESS\|p.process\_id = process\_id?} \]
\[ r! = \text{OK} \]

_StopProcess_ProcessIDNotExists_

\[ \text{\textsc{\small SYSTEM}} \]
\[ \text{process\_id\?: PROCESS\_ID} \]
\[ r! : \text{Response} \]
\[ \forall p : \text{processlist \bullet p.process\_id \neq process\_id?} \]
\[ r! = \text{ProcessIDNotExists} \]

\[ \text{StopProcess} = (\text{Authenticate\_User} \land (\text{StopProcess\_OK} \lor \text{StopProcess\_ProcessIDNotExists})) \lor \text{Authenticate\_User\_Fail} \]

.ChangeProcess_OK_

\[ \text{\textsc{\small SYSTEM}} \]
\[ \text{process\_id\?: PROCESS\_ID} \]
\[ \text{newstate\?: PROCESS\_STATE} \]
\[ r! : \text{Response} \]
\[ (\text{let } p = \{ p1 : \text{processlist | p1.process\_id = process\_id?} \} \bullet \]
\[ (\forall p2 : p \bullet p2.state = \text{newstate?}) \land \#p = 1 \land \]
\[ \text{processlist'} = \text{processlist\{p2 : \text{processlist | p2.process\_id = process\_id?} \} \land \}
\[ \text{processlist'} = \text{processlist'} \cup p \land \]
\[ r! = \text{OK} \]
\_ChangeProcess\_ProcessIDNotExists

\[ \exists \text{SYSTEM} \]
\[ \text{process\_id?} : \text{PROCESS\_ID} \]
\[ \text{newstate?} : \text{PROCESS\_STATE} \]
\[ r! : \text{Response} \]

\[ \forall p: \text{process\_list} \bullet p.\text{process\_id} \neq \text{process\_id}? \]
\[ r! = \text{ProcessIDNotExists} \]

\[ \text{ChangeProcess} \equiv (\text{Authenticate\_User} \land (\text{ChangeProcess\_OK} \lor \]
\[ \text{ChangeProcess\_ProcessIDNotExists})) \lor \text{Authenticate\_User\_Fail} \]
APPENDIX C

Authorization Tactic System Model Specification

[USER]
[FILENAME]
[FILE_PROPERTIES]
[PROCESS_ID]
[PROCESS_STATE]
Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|
            ProcessIDNotExists|IncorrectRights|UserNamesInconsistent|
            UserRunningProcess|FileBeingExecuted|UserStillAuthorized|
            FileExecutionStillAuthorized
UserAuthorization ::= Add_User|Delete_User
FileAuthorization ::= Add_File|Delete_File|Read_File|Change_File
ProcessAuthorization ::= Start_Process|Stop_Process|Change_Process
ACMAuthorization ::= User|File|Process|ACM

_PROCESS_
filename : FILENAME
owner : USER
process_id : PROCESS_ID
state : PROCESS_STATE
SYSTEM

filelist : FILENAME→FILE_PROPERTIES
userlist : USER
processlist : PROCESS
userACM : USER×USER→UserAuthorization
fileACM : USER×FILENAME→FileAuthorization
processACM : (USER×(USER×FILENAME))→ProcessAuthorization
acmACM : USER→PACMAuthorization

∀ p1, p2 : processlist • p1.process_id = p2.process_id ↔ p1 = p2
∀ p : processlist • p.filename ∈ dom filelist ∧ p.owner ∈ userlist.
dom (dom userACM) ⊆ userlist
dom (dom fileACM) ⊆ userlist
dom (dom processACM) ⊆ userlist
dom (ran (dom processACM)) ⊆ userlist
ran (ran (dom processACM)) ⊆ dom filelist
dom acmACM ⊆ userlist

InitSYSTEM

system : SYSTEM

#system.filelist = 1
#system.userlist = 1
#system.processlist = 1
#system.processACM = 1

AddACMACM_OK

ASYSTEM
user? : USER
newUser? : USER
rights? : PACMAuthorization
r! : Response

ACM ∈ acmACM (user?)
newUser? ∈ userlist
acmACM' = acmACM ∪ {newUser?→rights?}
r! = OK
\textit{AddACMACM\_UserNotExists}
\begin{align*}
\exists \text{SYSTEM} \\
user?: \text{USER} \\
\text{newUser}?: \text{USER} \\
\text{rights}?: \mathbb{P}\text{ACM}\text{Authorization} \\
r! : \text{Response} \\
ACM \in \text{acmACM}(user?) \\
\text{newUser} \notin \text{userlist} \\
r! = \text{UserNotExists}
\end{align*}

\textit{AddACMACM\_IncorrectRights}
\begin{align*}
\exists \text{SYSTEM} \\
user?: \text{USER} \\
\text{newUser}?: \text{USER} \\
\text{rights}?: \mathbb{P}\text{ACM}\text{Authorization} \\
r! : \text{Response} \\
ACM \notin \text{acmACM}(user?) \\
r! = \text{IncorrectRights}
\end{align*}

\textit{AddACMACM} \equiv \text{AddACMACM\_OK} \lor \text{AddACMACM\_UserNotExists} \lor \text{AddACMACM\_IncorrectRights}

\textit{DeleteACMACM\_OK}
\begin{align*}
\Delta \text{SYSTEM} \\
user?: \text{USER} \\
\text{deleteUser}?: \text{USER} \\
r! : \text{Response} \\
ACM \in \text{acmACM}(user?) \\
\text{deleteUser} \in \text{userlist} \\
\text{acmACM}' = \text{acmACM} \setminus \{\text{deleteUser}\rightarrow \text{acmACM}(\text{deleteUser})\} \\
r! = \text{OK}
\end{align*}
DeleteACMACM_UserNotExists

∃SYSTEM
user? : USER
deleteUser? : USER
r! : Response

ACM ∈ acmACM (user?)
deleteUser? ∉ userlist
r! = UserNotExists

DeleteACMACM_IncorrectRights

∃SYSTEM
user? : USER
deleteUser? : USER
r! : Response

ACM ∉ acmACM (user?)
r! = IncorrectRights

DeleteACMACM = DeleteACMACM_OK ∨ DeleteACMACM_UserNotExists ∨ DeleteACMACM_IncorrectRights

AddUserACM_OK

ΔSYSTEM
user? : USER
newUser? : USER
addUser? : USER
rights? : PUserAuthorization
r! : Response

User ∈ acmACM (user?)
newUser? ∈ userlist
userACM′ = userACM ∪ {(newUser?, addUser?)→rights?}
r! = OK
\_AddUserACM\_IncorrectRights

\[\exists \text{SYSTEM}
user? : \text{USER}
newUser? : \text{USER}
addUser? : \text{USER}
rights? : \text{P UserAuthorization}
\]

\[r! : \text{Response}\]

User e acmACM (user?)
\[r! = \text{IncorrectRights}\]

\_AddUserACM\_NewUserNotExists

\[\exists \text{SYSTEM}
user? : \text{USER}
newUser? : \text{USER}
addUser? : \text{USER}
rights? : \text{P UserAuthorization}
\]

\[r! : \text{Response}\]

User e acmACM (user?)
\[newUser? \notin \text{userlist}\]
\[r! = \text{UserNotExists}\]

\[\text{AddUserACM} \equiv \text{AddUserACM\_OK} \lor \text{AddUserACM\_IncorrectRights} \lor \]
\[\text{AddUserACM\_NewUserNotExists}\]
DeleteUserACM_OK

ΔSYSTEM
user? : USER
newUser? : USER
addUser? : USER
r! : Response

User ∈ acmACM (user?)
newUser? ∈ userlist
userACM = userACM \ {(newUser?, addUser?)→
    userACM(newUser?, addUser?)}

r! = OK

DeleteUserACMIncorrectRights

ΞSYSTEM
user? : USER
newUser? : USER
addUser? : USER
r! : Response

User ∈ acmACM (user?)

r! = IncorrectRights

DeleteUserACMNewUserNotExists

ΞSYSTEM
user? : USER
newUser? : USER
addUser? : USER
r! : Response

User ∈ acmACM (user?)
newUser? ∈ userlist

r! = UserNotExists
DeleteUserACM = DeleteUserACM_OK ∨ DeleteUserACM_IncorrectRights ∨ DeleteUserACM_NewUserNotExists

\_AddFileACM\_OK

\beginalphaSYSTEM
user? : USER
newUser? : USER
file? : FILENAME
rights? : PFileAuthorization
\endalpha
\beginbeta
File ∈ acmACM(user?)
newUser? ∈ userlist
fileACM = fileACM ∪ \{(newUser?, file?)→rights?\}
\endbeta
\beginepsilon
r! = OK
\endepsilon

\_AddFileACM\_IncorrectRights

\beginalphaSYSTEM
user? : USER
newUser? : USER
file? : FILENAME
rights? : PFileAuthorization
\endalpha
\beginbeta
File ∈ acmACM(user?)
\endbeta
\beginepsilon
r! = IncorrectRights
\endepsilon
AddFileACM_UserNotExists

\[ \exists \text{SYSTEM} \]
user? : USER
newUser? : USER
file? : FILENAME
rights? : FileAuthorization
r! : Response

File \in acmACM(user?)
newUser? \notin userlist
r! = UserNotExists

AddFileACM \equiv AddFileACM\_OK \lor AddFileACM\_IncorrectRights \lor AddFileACM\_UserNotExists

DeleteFileACM\_OK

\[ \Delta \text{SYSTEM} \]
user? : USER
newUser? : USER
file? : FILENAME
r! : Response

File \in acmACM(user?)
newUser? \in userlist
fileACM' = fileACM \setminus \{ (\text{newUser?}, file?) \rightarrow fileACM(\text{newUser?}, file?) \}

DeleteFileACM\_IncorrectRights

\[ \exists \text{SYSTEM} \]
user? : USER
newUser? : USER
file? : FILENAME
r! : Response

File \in acmACM(user?)
r! = IncorrectRights
DeleteFileACM _UserNotExists

\[ \exists \text{SYSTEM} \\
user? : \text{USER} \\
newUser? : \text{USER} \\
file? : \text{FILENAME} \\
r! : \text{Response} \\
\]

File \in \text{acmACM}(\text{user?})
newUser? \notin \text{userlist}
r! = \text{UserNotExists}

\[ \text{DeleteFileACM} \equiv \text{DeleteFileACM_OK} \lor \text{DeleteFileACM_IncorrectRights} \lor \text{DeleteFileACM_UserNotExists} \]

AddProcessACM_OK

\[ \Delta \text{SYSTEM} \\
user? : \text{USER} \\
newUser? : \text{USER} \\
runAs? : \text{USER} \\
file? : \text{FILENAME} \\
righs? : \text{PProcessAuthorization} \\
r! : \text{Response} \\
\]

Process \in \text{acmACM}(\text{user?})
newUser? \in \text{userlist}
processACM = processACM \cup \{(\text{newUser?},(\text{runAs?},\text{file?}))\rightarrow \text{rights?}\}
r! = \text{OK}
\( \text{AddProcessACM\_IncorrectRights} \)

\[ \Xi \text{SYSTEM} \]
\[ \text{user}?: \text{USER} \]
\[ \text{newUser}?: \text{USER} \]
\[ \text{runAs}?: \text{USER} \]
\[ \text{file}?: \text{FILENAME} \]
\[ \text{rights}?: \text{PProcessAuthorization} \]
\[ r! : \text{Response} \]

\[ \text{Process} \in \text{acmACM}(\text{user}) \]
\[ r! = \text{IncorrectRights} \]

\( \text{AddProcessACM\_UserNotExists} \)

\[ \Xi \text{SYSTEM} \]
\[ \text{user}?: \text{USER} \]
\[ \text{newUser}?: \text{USER} \]
\[ \text{runAs}?: \text{USER} \]
\[ \text{file}?: \text{FILENAME} \]
\[ \text{rights}?: \text{PProcessAuthorization} \]
\[ r! : \text{Response} \]

\[ \text{Process} \in \text{acmACM}(\text{user}) \]
\[ \text{newUser} \notin \text{userlist} \]
\[ r! = \text{UserNotExists} \]

\( \text{AddProcessACM} \equiv \text{AddProcessACM\_OK} \lor \text{AddProcessACM\_IncorrectRights} \lor \text{AddProcessACM\_UserNotExists} \)
DeleteProcessACM_OK

\Delta SYSTEM
user? : USER
newUser? : USER
runAs? : USER
file? : FILENAME
r! : Response

\[\text{Process} \in \text{acmACM (user?)}\]
\[\text{newUser?} \in \text{userlist}\]
\[\text{processACM} = \text{processACM} \setminus \{(\text{newUser?}, (\text{runAs?}, \text{file?}))\} \rightarrow \]
\[\text{processACM(\text{newUser?}, (\text{runAs?}, \text{file?}))}\]
\[r! = \text{OK}\]

DeleteProcessACM_IncorrectRights

\exists SYSTEM
user? : USER
newUser? : USER
runAs? : USER
file? : FILENAME
r! : Response

\[\text{Process} \in \text{acmACM (user?)}\]

r! = IncorrectRights

DeleteProcessACM_UserNotExists

\exists SYSTEM
user? : USER
newUser? : USER
runAs? : USER
file? : FILENAME
r! : Response

\[\text{Process} \in \text{acmACM (user?)}\]
\[\text{newUser?} \notin \text{userlist}\]

r! = UserNotExists
DeleteProcessACM ≡ DeleteProcessACM_OK \lor DeleteProcessACM_IncorrectRights \\
\lor DeleteProcessACM_UserNotExists

AddUser_Authorization_OK
\begin{align*}
\exists \text{SYSTEM} \\
\text{user?} : \text{USER} \\
\text{newUser?} : \text{USER} \\
\text{r!} : \text{Response} \\
\text{Add\_User} \in \text{userACM(user?,newUser?)} \\
\text{r!} = \text{OK}
\end{align*}

AddUser_Authorization_Fail
\begin{align*}
\exists \text{SYSTEM} \\
\text{user?} : \text{USER} \\
\text{newUser?} : \text{USER} \\
\text{r!} : \text{Response} \\
\text{Add\_User} \in \text{userACM(user?,newUser?)} \\
\text{r!} = \text{IncorrectRights}
\end{align*}

DeleteUser_Authorization_OK
\begin{align*}
\exists \text{SYSTEM} \\
\text{user?} : \text{USER} \\
\text{newUser?} : \text{USER} \\
\text{r!} : \text{Response} \\
\text{Delete\_User} \in \text{userACM(user?,newUser?)} \\
\text{r!} = \text{OK}
\end{align*}
\_DeleteUser\_Authorization\_Fail

\[\exists \text{SYSTEM}
user\? : \text{USER}
newUser\? : \text{USER}
 r! : \text{Response}
\]
\[\text{Delete}\_\text{User} \notin \text{userACM(user?,newUser?)}
 r! = \text{IncorrectRights}\]

\_AddFile\_Authorization\_OK

\[\exists \text{SYSTEM}
user\? : \text{USER}
file\? : \text{FILENAME}
 r! : \text{Response}
\]
\[\text{Add}\_\text{File} \in \text{fileACM(user?,file?)}
 r! = \text{OK}\]

\_AddFile\_Authorization\_Fail

\[\exists \text{SYSTEM}
user\? : \text{USER}
file\? : \text{FILENAME}
 r! : \text{Response}
\]
\[\text{Add}\_\text{File} \in \text{fileACM(user?,file?)}
 r! = \text{IncorrectRights}\]

\_DeleteFile\_Authorization\_OK

\[\exists \text{SYSTEM}
user\? : \text{USER}
file\? : \text{FILENAME}
 r! : \text{Response}
\]
\[\text{Delete}\_\text{File} \in \text{fileACM(user?,file?)}
 r! = \text{OK}\]
DeleteFile_Authorization_Fail

\[ \text{\texttt{SYSTEM}} \]
\[ \text{user? : USER} \]
\[ \text{file? : FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{Delete\_File} \in \text{fileACM}(\text{user?}, \text{file?}) \]
\[ r! = \text{IncorrectRights} \]

ReadFile_Authorization_OK

\[ \text{\texttt{SYSTEM}} \]
\[ \text{user? : USER} \]
\[ \text{file? : FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{Read\_File} \in \text{fileACM}(\text{user?}, \text{file?}) \]
\[ r! = \text{OK} \]

ReadFile_Authorization_Fail

\[ \text{\texttt{SYSTEM}} \]
\[ \text{user? : USER} \]
\[ \text{file? : FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{Read\_File} \in \text{fileACM}(\text{user?}, \text{file?}) \]
\[ r! = \text{IncorrectRights} \]

ChangeFile_Authorization_OK

\[ \text{\texttt{SYSTEM}} \]
\[ \text{user? : USER} \]
\[ \text{file? : FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{Change\_File} \in \text{fileACM}(\text{user?}, \text{file?}) \]
\[ r! = \text{OK} \]
ChangeFile_Authorization_Fail

\[ \exists \text{SYSTEM} \]
user? : USER
file? : FILENAME
r! : Response

\[ \text{Change\_File} \in \text{fileACM(user?, file?)} \]
\[ r! = \text{IncorrectRights} \]

StartProcess_Authorization_OK

\[ \exists \text{SYSTEM} \]
user? : USER
runAs? : USER
file? : FILENAME
r! : Response

\[ \text{Start\_Process} \in \text{processACM(user?, (runAs?, file?))} \]
\[ r! = \text{OK} \]

StartProcess_Authorization_Fail

\[ \exists \text{SYSTEM} \]
user? : USER
runAs? : USER
file? : FILENAME
r! : Response

\[ \text{Start\_Process} \in \text{processACM(user?, (runAs?, file?))} \]
\[ r! = \text{IncorrectRights} \]
StopProcess_Authorization_OK

\[ \exists \text{SYSTEM} \]
\[ \text{user?} : \text{USER} \]
\[ \text{runAs?} : \text{USER} \]
\[ \text{file?} : \text{FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{Stop\_Process} \in \text{processACM}(\text{user?}, (\text{runAs?}, \text{file?})) \]
\[ r! = \text{OK} \]

StopProcess_Authorization_Fail

\[ \exists \text{SYSTEM} \]
\[ \text{user?} : \text{USER} \]
\[ \text{runAs?} : \text{USER} \]
\[ \text{file?} : \text{FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{Stop\_Process} \in \text{processACM}(\text{user?}, (\text{runAs?}, \text{file?})) \]
\[ r! = \text{IncorrectRights} \]

ChangeProcess_Authorization_OK

\[ \exists \text{SYSTEM} \]
\[ \text{user?} : \text{USER} \]
\[ \text{runAs?} : \text{USER} \]
\[ \text{file?} : \text{FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{Change\_Process} \in \text{processACM}(\text{user?}, (\text{runAs?}, \text{file?})) \]
\[ r! = \text{OK} \]
\_ChangeProcess\_Authorization\_Fail

$$\exists \text{SYSTEM}
\begin{align*}
\text{user} &: \text{USER} \\
\text{runAs} &: \text{USER} \\
\text{file} &: \text{FILENAME} \\
\text{r!} &: \text{Response}
\end{align*}
$$

$\text{Change\_Process} \in \text{processACM}(\text{user?}, (\text{runAs?}, \text{file?}))$

$\text{r!} = \text{IncorrectRights}$

AddUser\_OK

$$\Delta \text{SYSTEM}
\begin{align*}
\text{newUser} &: \text{USER} \\
\text{r!} &: \text{Response}
\end{align*}
$$

$\text{newUser} \in \text{userlist}$

$\text{userlist'} = \text{userlist} \cup \{\text{newUser}\}$

$\text{r!} = \text{OK}$

AddUser\_Exists

$$\exists \text{SYSTEM}
\begin{align*}
\text{newUser} &: \text{USER} \\
\text{r!} &: \text{Response}
\end{align*}
$$

$\text{newUser} \in \text{userlist}$

$\text{r!} = \text{UserExists}$

AddUser \equiv (AddUser\_Authorization\_OK \land (AddUser\_OK \lor AddUser\_Exists)) \lor (AddUser\_Authorization\_Fail)$
**Delete User OK**

\text{\Delta SYSTEM}
\text{deleteUser? : USER}
\text{r! : Response}

\text{deleteUser? \in userlist}
\forall p : \text{processlist} \ni p.\text{owner} \neq \text{deleteUser?}
\text{deleteUser? \in dom (dom userACM)}
\text{deleteUser? \in dom (dom fileACM)}
\text{deleteUser? \in dom (dom processACM)}
\text{deleteUser? \in dom (ran (dom processACM))}
\text{deleteUser? \in dom acmACM}
\text{userlist' = userlist \setminus \{ deleteUser?\}}
\text{r! = OK}

**Delete User NotExists**

\text{\Xi SYSTEM}
\text{deleteUser? : USER}
\text{r! : Response}

\text{deleteUser? \notin userlist}
\text{r! = UserNotExists}

**Delete User RunningProcess**

\text{\Xi SYSTEM}
\text{deleteUser? : USER}
\text{r! : Response}

\text{deleteUser? \in userlist}
\exists p : \text{processlist} \ni p.\text{owner} = \text{deleteUser?}
\text{r! = UserRunningProcess}
DeleteUser - UserStillAuthorized

$\exists$SYSTEM

deleteUser? : USER

r! : Response

deleteUser? $\in$ userlist

$\forall p : processlist \in p.\text{owner} \neq$ deleteUser?

deleteUser? $\in$ dom (dom userACM) $\lor$

deleteUser? $\in$ dom (dom fileACM) $\lor$

deleteUser? $\in$ dom (dom processACM) $\lor$

deleteUser? $\in$ dom (ran (dom processACM)) $\lor$

deleteUser? $\in$ dom acmACM

r! = UserStillAuthorized

DeleteUser $\equiv$ (DeleteUser_Authorization_OK $\land$ (DeleteUser_OK $\lor$

DeleteUser_NotExists $\lor$ DeleteUser_RunningProcess $\lor$

DeleteUser_UserStillAuthorized)) $\lor$ DeleteUser_Authorization_Fail

AddFile_OK

$\Delta$SYSTEM

filename? : FILENAME

fileproperties? : FILE_PROPERTIES

r! : Response

filename? $\in$ dom filelist

filelist' = filelist $\cup$ \{filename? $\rightarrow$ fileproperties?\}

r! = OK
AddFile Exists

\[ \exists \text{SYSTEM} \]
filename? : FILENAME
fileproperties? : FILE_PROPERTIES
r! : Response

filename? \in dom filelist
r! = FileExists

AddFile \equiv (AddFile\_Authorization\_OK \land (AddFile\_OK \lor AddFile\_Exists)) \lor AddFile\_Authorization\_Fail

ReadFile OK

\[ \exists \text{SYSTEM} \]
filename? : FILENAME
fileproperties! : FILE_PROPERTIES
r! : Response

filename? \in dom filelist
fileproperties! = filelist(filename?)
r! = OK

ReadFile NotExists

\[ \exists \text{SYSTEM} \]
filename? : FILENAME
fileproperties! : FILE_PROPERTIES
r! : Response

filename? \notin dom filelist
r! = FileNotExists

ReadFile \equiv (ReadFile\_Authorization\_OK \land (ReadFile\_OK \lor ReadFile\_NotExists)) \lor ReadFile\_Authorization\_Fail
DeleteFile_OK

ΔSYSTEM
filename? : FILENAME
r! : Response

filename? ∈ dom filelist
∀p : processlist • p.filename ≠ filename?
filename? ∈ ran (ran (dom processACM))
filelist' = filelist \ {filename?→ filelist(filename?)}
r! = OK

DeleteFile_NotExists

ΞSYSTEM
filename? : FILENAME
r! : Response

filename? ∈ dom filelist
r! = FileNotExists

DeleteFile_FileBeingExecuted

ΞSYSTEM
filename? : FILENAME
r! : Response

filename? ∈ dom filelist
∃p : processlist • p.filename = filename?
r! = FileBeingExecuted
\_DeleteFile\_ FileExecutionStillAuthorized

\[ \exists \text{SYSTEM} \]
\[ \text{filename}\? : \text{FILENAME} \]
\[ r! : \text{Response} \]
\[ \text{filename}\? \in \text{dom filelist} \]
\[ \forall p : \text{processlist} \bullet p.\text{filename} \neq \text{filename}\? \]
\[ \text{filename}\? \in \text{ran (ran (dom process\_ACM))} \]
\[ r! = \text{FileExecutionStillAuthorized} \]

\[ \text{DeleteFile} \equiv (\text{DeleteFile\_Authorization\_OK} \land (\text{DeleteFile\_OK} \lor \]
\[ \phantom{\text{DeleteFile}} \text{DeleteFile\_NotExists} \lor \text{DeleteFile\_FileBeingExecuted} \lor \]
\[ \phantom{\text{DeleteFile}} \text{DeleteFile\_FileExecutionStillAuthorized}) \lor \]
\[ \phantom{\text{DeleteFile}} \text{DeleteFile\_Authorization\_Fail} \]

\_ChangeFile\_ OK

\[ \Delta \text{SYSTEM} \]
\[ \text{filename}\? : \text{FILENAME} \]
\[ \text{properties}\? : \text{FILE\_PROPERTIES} \]
\[ r! : \text{Response} \]
\[ \text{filename}\? \in \text{dom filelist} \]
\[ \text{filelist}' = \text{filelist} \oplus \{\text{filename}\? \mapsto \text{properties}\?\} \]
\[ r! = \text{OK} \]

\_ChangeFile\_ NotExists

\[ \exists \text{SYSTEM} \]
\[ \text{filename}\? : \text{FILENAME} \]
\[ \text{properties}\? : \text{FILE\_PROPERTIES} \]
\[ r! : \text{Response} \]
\[ \text{filename}\? \in \text{dom filelist} \]
\[ r! = \text{FileNotExists} \]
\[
\text{ChangeFile} \equiv (\text{ChangeFile\_Authorization\_OK} \land (\text{ChangeFile\_OK} \lor \\
\text{ChangeFile\_NotExists})) \lor \text{ChangeFile\_Authorization\_Fail}
\]

\[\]

\text{StartProcess\_OK}\]

ΔSYSTEM
process? : PROCESS
runAs? : USER
r! : Response

\[
\text{process?.owner} = \text{runAs}?
\]
\[
\text{process?.filename} \in \text{dom\_filelist}
\]
\[
\forall p: \text{processlist} \bullet p.\text{process\_id} = \text{process?.process\_id}
\]
\[
\text{processlist}' = \text{processlist} \cup \{\text{process?}\}
\]
\[
r! = \text{OK}
\]

\text{StartProcess\_FileNotExists}\]

ΞSYSTEM
process? : PROCESS
runAs? : USER
r! : Response

\[
\text{process?.filename} \notin \text{dom\_filelist}
\]
\[
r! = \text{FileNotExists}
\]

\text{StartProcess\_UserNamesInconsistent}\]

ΞSYSTEM
process? : PROCESS
runAs? : USER
r! : Response

\[
\text{process?.owner} \neq \text{runAs}?
\]
\[
r! = \text{UserNamesInconsistent}
\]
\[ \text{StartProcess} \equiv (\text{StartProcess\_Authorization\_OK} \land (\text{StartProcess\_OK} \lor \\
\text{StartProcess\_FileNotExists} \lor \text{StartProcess\_UserNameInconsistent})) \lor \\
\text{StartProcess\_Authorization\_Fail} \]

\text{StopProcess\_OK}
\[
\Delta \text{SYSTEM} \\
\text{process\_id}\& : \text{PROCESS\_ID} \\
\text{runAs}\& : \text{USER} \\
r! : \text{Response} \\
\exists p: \text{processlist} \bullet p.\text{process\_id} = \text{process\_id}\& \land p.\text{owner} = \text{runAs}\& \\
\text{processlist} = \text{processlist}\{p: \text{PROCESS}\{p.\text{process\_id} = \text{process\_id}\&\}\} \\
r! = \text{OK} \\
\]

\text{StopProcess\_ProcessIDNotExists}
\[
\exists \text{SYSTEM} \\
\text{process\_id}\& : \text{PROCESS\_ID} \\
\text{runAs}\& : \text{USER} \\
r! : \text{Response} \\
\forall p: \text{processlist} \bullet p.\text{process\_id} \neq \text{process\_id}\& \\
r! = \text{ProcessIDNotExists} \\
\]

\text{StopProcess\_UserNameInconsistent}
\[
\exists \text{SYSTEM} \\
\text{process\_id}\& : \text{PROCESS\_ID} \\
\text{runAs}\& : \text{USER} \\
r! : \text{Response} \\
\exists p: \text{processlist} \bullet p.\text{process\_id} = \text{process\_id}\& \\
\forall p: \text{processlist} \bullet p.\text{owner} \neq \text{runAs}\& \\
r! = \text{UserNameInconsistent} \\
\]
StopProcess ≡ (StopProcess_Authorization_OK ∧ (StopProcess_OK ∨ StopProcess_ProcessIDNotExists ∨ StopProcess_UserNamesInconsistent)) ∨ StopProcess_Authorization_Fail

ChangeProcess_OK

\[ \Delta \text{SYSTEM} \]

\begin{align*}
\text{process_id}? &: \text{PROCESS_ID} \\
\text{newstate}? &: \text{PROCESS_STATE} \\
r! &: \text{Response} \\
(\text{let } p &= \{p_1: \text{processlist} | p_1.\text{process_id} = \text{process_id}\} ∗ \\
(\forall p_2: p ∗ p_2.\text{state} = \text{newstate}?) ∧ \#p = 1 ∨ \\
\text{processlist'} = \text{processlist'} \setminus \{p_2: \text{processlist} | p_2.\text{process_id} = \text{process_id}\} ∧ \\
\text{processlist'} = \text{processlist'} ∪ p ∨ \\
r! = \text{OK})
\end{align*}

ChangeProcess_ProcessIDNotExists

\[ \Xi \text{SYSTEM} \]

\begin{align*}
\text{process_id}? &: \text{PROCESS_ID} \\
\text{newstate}? &: \text{PROCESS_STATE} \\
r! &: \text{Response} \\
(\forall p: \text{processlist} ∗ p.\text{process_id} ≠ \text{process_id}) \\
r! = \text{ProcessIDNotExists}
\end{align*}

APPENDIX D

Data Confidentiality Tactic System Model Specification

[USER]
[F I L E _ N A M E]
[F I L E _ P R O P E R T I E S]
[P R O C E S S _ I D]
[P R O C E S S _ S T A T E]
[S E C R E T]

Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|
            ProcessIDNotExists|InvalidFileSecretPair|FileExists_Different|
            UserRunningProcess|FileBeingExecuted

PROCESS

filename : FILENAME
owner : USER
process_id : PROCESS_ID
state : PROCESS_STATE

SYSTEM

filelist : FILENAME × SECRET → FILE_PROPERTIES
userlist : P USER
processlist : P PROCESS

∀p1, p2 : processlist • p1.process_id = p2.process_id ↔ p1 = p2
∀p : processlist • p.filename ∈ dom(filelist) ∧ p.owner ∈ userlist
∀f : dom(filelist) • ∀s1, s2 : (dom(filelist) union {f}) • filelist(f, s1) = filelist(f, s2)
\textit{Init}SYSTEM

\begin{align*}
\text{system} & : \text{SYSTEM} \\
\#\text{system.filelist} &= 1 \\
\#\text{system.userlist} &= 1 \\
\#\text{system.processlist} &= 1
\end{align*}

\textit{AddUser\_OK}

\begin{align*}
\Delta\text{SYSTEM} \\
\text{newUser?} & : \text{USER} \\
r! & : \text{Response} \\
\text{newUser?} & \in \text{userlist} \\
\text{userlist}' &= \text{userlist} \cup \{\text{newUser?}\} \\
r! &= \text{OK}
\end{align*}

\textit{AddUser\_Exists}

\begin{align*}
\Xi\text{SYSTEM} \\
\text{newUser?} & : \text{USER} \\
r! & : \text{Response} \\
\text{newUser?} & \in \text{userlist} \\
r! &= \text{UserExists}
\end{align*}

\textit{AddUser} \equiv \textit{AddUser\_OK} \lor \textit{AddUser\_Exists}

\textit{DeleteUser\_OK}

\begin{align*}
\Delta\text{SYSTEM} \\
\text{user?} & : \text{USER} \\
r! & : \text{Response} \\
\text{user?} & \in \text{userlist} \\
\forall p : \text{processlist} \bullet p.\text{owner} \neq \text{user?} \\
\text{userlist}' &= \text{userlist} \setminus \{\text{user?}\} \\
r! &= \text{OK}
\end{align*}
DeleteUser_NotExists

\[ \exists \text{SYSTEM} \\
\text{user? : USER} \\
\text{r! : Response} \]
\text{user? \notin \text{userlist}} \\
r! = \text{UserNotExists}

DeleteUser_RunningProcess

\[ \exists \text{SYSTEM} \\
\text{user? : USER} \\
\text{r! : Response} \]
\text{user? \notin \text{userlist}} \\
\exists p : \text{processlist} \bullet p.\text{owner} = \text{user?} \\
r! = \text{UserRunningProcess}

DeleteUse \equiv \text{DeleteUser_OK} \lor \text{DeleteUser_NotExists} \lor \text{DeleteUser_RunningProcess}

AddFile_OK

\[ \nabla \text{SYSTEM} \\
\text{filename? : FILENAME} \\
\text{secret? : SECRET} \\
\text{fileproperties? : FILE_PROPERTIES} \\
\text{r! : Response} \]
\text{(filename?,secret?) \notin \text{dom filelist}} \\
\text{filename? \notin \text{dom (dom filelist)} \lor (\forall s : (\text{dom filelist}) \{(\text{filename?})\} \bullet \\
\text{fileproperties? = filelist(filename?,s))} \\
\text{filelist' = filelist \cup \{(filename?,secret?) \rightarrow fileproperties?)} \\
r! = \text{OK} \]
\textit{AddFile\_Exists}\_
\texttt{\$SYSTEM$
filename\? : FILENAME
secret\? : SECRET
fileproperties\? : FILE\_PROPERTIES
r! : Response

\((\text{filename}\?, \text{secret}\?) \in \text{dom filelist})$
\ r! = \text{FileExists}$

\textit{AddFile\_Exists\_Diff}\_
\texttt{\$SYSTEM$
filename\? : FILENAME
secret\? : SECRET
fileproperties\? : FILE\_PROPERTIES
r! : Response

\((\text{filename}\?) \in \text{dom (dom filelist)}$
\ \ \forall s : (\text{dom filelist}) \ (\{\text{filename}\?\}) \bullet s \neq \text{secret}\?$
\ \ \exists s : (\text{dom filelist}) \ (\{\text{filename}\?\}) \bullet \text{filelist(filename}\?, s) \neq \text{fileproperties}\?
\ r! = \text{FileExists\_Different}$

\text{AddFile} \equiv \text{AddFile\_OK} \lor \text{AddFile\_Exists} \lor \text{AddFile\_Exists\_Diff}$

\textit{ReadFile\_OK}\_
\texttt{\$SYSTEM$
filename\? : FILENAME
secret\? : SECRET
fileproperties\!: FILE\_PROPERTIES
r! : Response

\((\text{filename}\?, \text{secret}\?) \in \text{dom filelist}$
\text{fileproperties}\! = \text{filelist(filename}\?, \text{secret}\?)$
\ r! = \text{OK}$
ReadFile NotExists

\[
\text{\textbf{ReadFile NotExists}} \\
\exists \text{SYSTEM} \\
\text{filename?): FILENAME} \\
\text{secret?): SECRET} \\
\text{fileproperties!: FILE_PROPERTIES} \\
r!: \text{Response} \\
\text{(filename?,secret?) } \notin \text{ dom filelist} \\
r! = \text{InvalidFileSecretPair}
\]

ReadFile \equiv \text{ReadFile OK} \lor \text{ReadFile NotExists}

DeleteFile_OK

\[
\Delta \text{SYSTEM} \\
\text{filename?): FILENAME} \\
\text{secret?): SECRET} \\
r!: \text{Response} \\
\text{(filename?,secret?) } \in \text{ dom filelist} \\
\forall p : \text{processlist} \bullet p_{filename} \neq \text{filename?} \\
\text{(let secretlist } = (\text{dom filelist}) \{\{\text{filename?}\}\}) \bullet \\
\text{filelist'} = \text{filelist} \setminus (\{\text{filename?}\} \times \text{secretlist}) \triangle \text{filelist} \\
r! = \text{OK}
\]

DeleteFile_NotExists

\[
\exists \text{SYSTEM} \\
\text{filename?): FILENAME} \\
\text{secret?): SECRET} \\
r!: \text{Response} \\
\text{(filename?,secret?) } \in \text{ dom filelist} \\
r! = \text{InvalidFileSecretPair}
\]
DeleteFile _ FileBeingExecuted

\[ \exists \text{SYSTEM} \]
filename? : FILENAME
secret? : SECRET
r! : Response

\[(\text{filename?}, \text{secret?}) \in \text{dom filelist} \]
\[ \exists p : \text{processlist} \cdot p. \text{filename} = \text{filename}? \]
r! = FileBeingExecuted

DeleteFile \equiv \text{DeleteFile}_\text{OK} \lor \text{DeleteFile}_\text{NotExists} \lor 
\text{DeleteFile}_\text{FileBeingExecuted}

ChangeFile _ OK

\[ \Delta \text{SYSTEM} \]
filename? : FILENAME
secret? : SECRET
properties? : FILE_PROPERTIES
r! : Response

\[(\text{filename?}, \text{secret?}) \in \text{dom filelist} \]
\[(\text{let secretlist} = ((\text{dom filelist}) \{\text{filename}?\})) \cdot \]
\[\text{filelist}' = \text{filelist} \oplus ((\{\text{filename}?\} \times \text{secretlist}) \times \{\text{properties}?\})) \]
r! = OK

ChangeFile _ NotExists

\[ \exists \text{SYSTEM} \]
filename? : FILENAME
secret? : SECRET
properties? : FILE_PROPERTIES
r! : Response

\[(\text{filename?}, \text{secret?}) \in \text{dom filelist} \]
r! = InvalidFileSecretPair
ChangeFile ≡ ChangeFile_OK ∨ ChangeFile_NotExists

\[
\text{StartProcess_OK} \quad \Delta \text{SYSTEM} \\
\text{process? : PROCESS} \\
\text{secret? : SECRET} \\
\text{r! : Response} \\
\text{process?.owner ∈ userlist} \\
(\text{process?.filename, secret?}) ∈ \text{dom filelist} \\
\forall p: \text{processlist} \; p.\text{process_id} ≠ \text{process?.process_id} \\
\text{processlist'} = \text{processlist} \cup \{\text{process?}\} \\
r! = \text{OK}
\]

\[
\text{StartProcess_FileNotExists} \quad \Xi \text{SYSTEM} \\
\text{process? : PROCESS} \\
\text{secret? : SECRET} \\
\text{r! : Response} \\
(\text{process?.filename, secret?}) ∈ \text{dom filelist} \\
r! = \text{InvalidFileSecretPair}
\]

\[
\text{StartProcess_UserNotExists} \quad \Xi \text{SYSTEM} \\
\text{process? : PROCESS} \\
\text{r! : Response} \\
\text{process?.owner ∈ userlist} \\
r! = \text{UserNotExists}
\]

StopProcess_OK

system
process_id? : PROCESS_ID
r! : Response

\exists p:processlist \bullet p.process_id = process_id?
processlist' = processlist\{p:PROCESS\{p.process_id = process_id?\}
\r! = OK

StopProcess_ProcessIDNotExists

\exists system
process_id? : PROCESS_ID
r! : Response

\forall p:processlist \bullet p.process_id \neq process_id?
\r! = ProcessIDNotExists

StopProcess \equiv StopProcess_OK \lor StopProcess_ProcessIDNotExists

ChangeProcess_OK

system
process_id? : PROCESS_ID
newstate? : PROCESS_STATE
r! : Response

(let p = \{p1: processlist \mid p1.process_id = process_id?\} •
(\forall p2: p \bullet p2.state = newstate?) \#p = 1 \land
processlist' = processlist\{p2: processlist \mid p2.process_id = process_id?\} \land
processlist' = processlist\cup p \land
\r! = OK)
\texttt{ChangeProcess\_ProcessIDNotExists}\\
\exists \text{SYSTEM}\\
\text{process\_id? : PROCESS\_ID}\\
\text{newstate? : PROCESS\_STATE}\\
\text{r! : Response}\\
\forall p: \text{processlist} \bullet p.\text{process\_id} \neq \text{process\_id}?\\
\text{r! = ProcessIDNotExists}\\

\texttt{ChangeProcess} \equiv \texttt{ChangeProcess\_OK} \lor \texttt{ChangeProcess\_ProcessIDNotExists}
APPENDIX E

Data Integrity Tactic System Model Specification

[USER]
[FILENAME]
[FILE_INFO]
[INTEGRITY_CHECK]
[PROCESS_ID]
[PROCESS_STATE]
[INTEGRITY_CONTROL]

Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|
            ProcessIDNotExists|InvalidIntegrityCheck|UserRunningProcess
            |FileBeingExecuted

\[ \text{INTEGRITY\_MAP} : \text{FILE\_INFO} \times \text{INTEGRITY\_CONTROL} \rightarrow \text{INTEGRITY\_CHECK} \]

\[ \text{FILE\_PROPERTIES} \]
\[ \text{info} : \text{FILE\_INFO} \]
\[ \text{integrity} : \text{INTEGRITY\_CHECK} \]
\[ \exists c : \text{INTEGRITY\_CONTROL} \cdot \text{INTEGRITY\_MAP}(\text{info}, c) = \text{integrity} \]

\[ \text{PROCESS} \]
\[ \text{filename} : \text{FILENAME} \]
\[ \text{owner} : \text{USER} \]
\[ \text{process\_id} : \text{PROCESS\_ID} \]
\[ \text{state} : \text{PROCESS\_STATE} \]
**SYSTEM**

filelist : FILENAME → FILE_PROPERTIES
userlist : USER
processlist : PROCESS

∀p1, p2 : processlist • p1.process_id = p2.process_id ↔ p1 = p2
∀p : processlist • p.filename ∈ dom filelist ∧ p.owner ∈ userlist

**Check Integrity**

∃SYSTEM

filename? : FILENAME
ic? : INTEGRITY_CONTROL

filename? ∈ dom (filelist)
INTEGRITY_MAP((filelist(filename?)).info, ic?) = (filelist(filename?)).integrity

**Check Integrity Fail**

∃SYSTEM

filename? : FILENAME
ic? : INTEGRITY_CONTROL
r! : Response

filename? ∈ dom (filelist)
INTEGRITY_MAP((filelist(filename?)).info, ic?) ≠ (filelist(filename?)).integrity
r! = InvalidIntegrityCheck

**InitSYSTEM**

system : SYSTEM

#system.filelist = 1
#system.userlist = 1
#system.processlist = 1
**AddUser_OK**

\[
\begin{align*}
\text{SYSTEM} \\
\text{newUser? : USER} \\
\text{r! : Response} \\
\text{newUser? \notin \text{userlist}} \\
\text{userlist'} = \text{userlist} \cup \{\text{newUser?}\} \\
\text{r! = OK}
\end{align*}
\]

**AddUser_Exists**

\[
\begin{align*}
\text{SYSTEM} \\
\text{newUser? : USER} \\
\text{r! : Response} \\
\text{newUser? \in \text{userlist}} \\
\text{r! = UserExists}
\end{align*}
\]

**AddUser \equiv AddUser_OK \lor AddUser_Exists**

**DeleteUser_OK**

\[
\begin{align*}
\text{SYSTEM} \\
\text{user? : USER} \\
\text{r! : Response} \\
\text{user? \in \text{userlist}} \\
\forall p : \text{processlist} \cdot p.\text{owner} \neq \text{user?} \\
\text{userlist'} = \text{userlist} \setminus \{\text{user?}\} \\
\text{r! = OK}
\end{align*}
\]
DeleteUser_NotExists

\[\exists SYSTEM\]
user? : USER
r! : Response

user? \notin userlist
r! = UserNotExists

DeleteUser_RunningProcess

\[\exists SYSTEM\]
user? : USER
r! : Response

user? \in userlist
\exists p : processlist \bullet p.owner = user?
\quad r! = UserRunningProcess

DeleteUse \equiv DeleteUser_OK \lor DeleteUser_NotExists \lor DeleteUser_RunningProcess

AddFile_OK

\[\Delta SYSTEM\]
filename? : FILENAME
fileproperties? : FILE_PROPERTIES
r! : Response

filename? \in dom filelist
\exists ic : INTEGRITY_CONTROL \bullet INTEGRITY_MAP(fileproperties?.info,ic)
\qquad = fileproperties?.integrity
filelist' = filelist \cup \{filename? \rightarrow fileproperties?\}
r! = OK
AddFile_Exists

∀SYSTEM
filename? : FILENAME
fileproperties? : FILE_PROPERTIES
r! : Response

filename? ∈ dom filelist
r! = FileExists

AddFile_Integrity_Fail

∀SYSTEM
filename? : FILENAME
fileproperties? : FILE_PROPERTIES
r! : Response

filename? ∈ dom filelist
¬(∃ic : INTEGRITY_CONTROL • INTEGRITY_MAP(fileproperties?.info.ic)
   = fileproperties?.integrity)

r! = InvalidIntegrityCheck

AddFile ≡ AddFile_OK ∨ AddFile_Exists ∨ AddFile_Integrity_Fail

ReadFile_OK

∀SYSTEM
Check_Integrity
filename? : FILENAME
fileproperties! : FILE_PROPERTIES
r! : Response

filename? ∈ dom filelist
fileproperties! = filelist(filename?)

r! = OK
_ReadFile_NotExists_

\[\exists \text{SYSTEM} \]
\[\text{filename}? : \text{FILENAME} \]
\[\text{fileproperties!} : \text{FILE\_PROPERTIES} \]
\[r! : \text{Response} \]
\[\text{filename}? \in \text{dom filelist} \]
\[r! = \text{FileNotExists} \]

\(\text{ReadFile} \equiv \text{ReadFile\_OK} \lor \text{ReadFile\_NotExists} \lor \text{Check\_Integrity\_Fail} \)

_DeleteFile_OK_

\[\Delta \text{SYSTEM} \]
\[\text{filename}? : \text{FILENAME} \]
\[r! : \text{Response} \]
\[\text{filename}? \in \text{dom filelist} \]
\[\forall p : \text{processlist} \bullet p.\text{filename} \neq \text{filename}? \]
\[\text{filelist'} = \text{filelist} \setminus \{\text{filename}?\rightarrow \text{filelist}(\text{filename}?)\} \]
\[r! = \text{OK} \]

_DeleteFile_NotExists_

\[\exists \text{SYSTEM} \]
\[\text{filename}? : \text{FILENAME} \]
\[r! : \text{Response} \]
\[\text{filename}? \in \text{dom filelist} \]
\[r! = \text{FileNotExists} \]
_DeleteFile_FileBeingExecuted_

SYSYSTEM
filename?: FILENAME
r!: Response

filename? \in dom filelist
\exists p : processlist . p.filename = filename?
r! = FileBeingExecuted

_DeleteFile = DeleteFile_OK \lor DeleteFile_NotExists \lor DeleteFile_FileBeingExecuted_

_ChangeFile_OK_

\Delta SYSTEM
Check_Integrity
filename?: FILENAME
properties?: FILE_PROPERTIES
r!: Response

filename? \in dom filelist
\exists ic : INTEGRITY_CONTROL . INTEGRITY_MAP(properties?.info,ic)
= properties?.integrity
filelist' = filelist \oplus \{filename? \mapsto properties?\}
r! = OK

_ChangeFile_NotExists_

SYSYSTEM
filename?: FILENAME
properties?: FILE_PROPERTIES
r!: Response

filename? \in dom filelist
r! = FileNotExists
\_ChangeFile\_Integrity\_Fail

\(\exists \text{SYSTEM} \\\n\text{filename}\? : \text{FILENAME} \\\n\text{properties}\? : \text{FILE\_PROPERTIES} \\\nr! : \text{Response} \)

\(\text{filename}\? \in \text{dom filelist} \\\n\neg (\exists ic : \text{INTEGRITY\_CONTROL} \bullet \text{INTEGRITY\_MAP}(\text{properties}\?.\text{info}, ic) \\\n= \text{properties}\?.\text{integrity}) \\\n\text{filelist}' = \text{filelist} \cup \{\text{filename}\? \leftrightarrow \text{properties}\?\} \\\nr! = \text{InvalidIntegrityCheck} \)

\(\text{ChangeFile} \equiv \text{ChangeFile\_OK} \lor \text{ChangeFile\_NotExists} \lor \text{Check\_Integrity\_Fail} \lor \text{ChangeFile\_Integrity\_Fail} \)

\_StartProcess\_OK

\(\Delta \text{SYSTEM} \\\n\text{Check\_Integrity} \\\n\text{process}\? : \text{PROCESS} \\\nr! : \text{Response} \)

\(\text{process}\?.\text{owner} \in \text{userlist} \\\n\text{process}\?.\text{filename} \in \text{dom filelist} \\\n\forall p: \text{processlist} \bullet p.\text{process\_id} \neq \text{process}\?.\text{process\_id} \\\n\text{processlist}' = \text{processlist} \cup \{\text{process}\?\} \\\nr! = \text{OK} \)

\_StartProcess\_FileNotExists

\(\exists \text{SYSTEM} \\\n\text{process}\? : \text{PROCESS} \\\nr! : \text{Response} \)

\(\text{process}\?.\text{filename} \in \text{dom filelist} \\\nr! = \text{FileNotExists} \)
\[ \text{StartProcess\_UserNotExists} \]
\[
\exists \text{SYSTEM} \\
\text{process? : PROCESS} \\
\text{r! : Response} \\
\text{process?.owner } \notin \text{ userlist} \\
\text{r! = UserNotExists}
\]

\[ \text{StartProcess} \equiv \text{StartProcess\_OK} \lor \text{StartProcess\_FileNotExists} \lor \text{StartProcess\_UserNotExists} \lor \text{Check\_Integrity\_Fail} \]

\[ \text{StopProcess\_OK} \]
\[
\Delta \text{SYSTEM} \\
\text{Check\_Integrity} \\
\text{process_id? : PROCESS\_ID} \\
\text{r! : Response} \\
\exists p:\text{processlist} \bullet p.\text{process_id} = \text{process_id} \\
\text{processlist'} = \text{processlist} \setminus \{p:\text{PROCESS}\{p.\text{process_id} = \text{process_id}'\}\} \\
\text{r! = OK}
\]

\[ \text{StopProcess\_ProcessIDNotExists} \]
\[
\exists \text{SYSTEM} \\
\text{process_id? : PROCESS\_ID} \\
\text{r! : Response} \\
\forall p:\text{processlist} \bullet p.\text{process_id} \neq \text{process_id} \\
\text{r! = ProcessIDNotExists}
\]

\[ \text{StopProcess} \equiv \text{StopProcess\_OK} \lor \text{StopProcess\_ProcessIDNotExists} \lor \text{Check\_Integrity\_Fail} \]
\textit{ChangeProcess\_OK}

\textbf{SYSTEM}

\textbf{Check\_Integrity}

\textbf{process\_id}?: \texttt{PROCESS\_ID}

\textbf{newstate}?: \texttt{PROCESS\_STATE}

\textbf{r}!: \texttt{Response}

(let \( p = \{ p_1: \text{processlist} \mid p_1.\text{process\_id} = \text{process\_id}\} \bullet \)

(\( \forall p_2: p \bullet p_2.\text{state} = \text{newstate}\) \( \land \#p = 1 \land \)

\text{processlist'} = \text{processlist} \setminus \{ p_2: \text{processlist} \mid p_2.\text{process\_id} = \text{process\_id}\} \land \\

\text{processlist'} = \text{processlist'} \cup p \land \\

\textbf{r}! = \text{OK})

\textit{ChangeProcess\_ProcessIDNotExists}

\textbf{SYSTEM}

\textbf{process\_id}?: \texttt{PROCESS\_ID}

\textbf{newstate}?: \texttt{PROCESS\_STATE}

\textbf{r}!: \texttt{Response}

(\( \forall p: \text{processlist} \bullet p.\text{process\_id} \neq \text{process\_id}\) \\

\textbf{r}! = \text{ProcessIDNotExists}

\textit{ChangeProcess} \equiv \textit{ChangeProcess\_OK} \lor \textit{ChangeProcess\_ProcessIDNotExists} \\
\lor \textit{Check\_Integrity\_Fail}
APPENDIX F

Auditing Tactic System Model Specification

[USER]
[FILENAME]
[FILE_PROPERTIES]
[FILE_CHANGE]
[PROCESS_STATE]
[PROCESS_STATE_CHANGE]
[PROCESS_ID]

UserAction ::= Add_User|Delete_User
FileAction ::= Add_File|Delete_File|Read_File|Change_File
ProcessAction ::= Start_Process|Stop_Process|Change_Process
Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|
                     ProcessIDNotExists|UserRunningProcess|FileBeingExecuted

FILE_CHANGE_MAP : FILE_PROPERTIES × FILE_CHANGE
                      → FILE_PROPERTIES

FILE_PROPERTIES_MAP : FILE_PROPERTIES × FILE_PROPERTIES × FILE_PROPERTIES
                        → FILE_CHANGE

PROCESS_STATE_CHANGE_MAP : PROCESS_STATE × PROCESS_STATE
                             PROCESS_STATE_CHANGE → PROCESS_STATE

PROCESS_STATE_MAP : PROCESS_STATE × PROCESS_STATE
                     → PROCESS_STATE_CHANGE
UserLogElement
user: USER
action: UserAction

FileLogElement
filename: FILENAME
action: FileAction
filechange: FILE_CHANGE

ProcessLogElement
process_id: PROCESS_ID
filename: FILENAME
owner: USER
action: ProcessAction
processchange: PROCESS_STATE_CHANGE

LogElement
u: UserLogElement
f: FileLogElement
p: ProcessLogElement
r : Response

#u + #f + #p = 1

PROCESS
filename: FILENAME
owner: USER
process_id: PROCESS_ID
state: PROCESS_STATE
SYSTEM

filelist : FILENAME → FILE_PROPERTIES
userlist : P USER
processlist : P PROCESS
userauditpolicy : P(USER × UserAction)
fileauditpolicy : P(FILENAME × FileAction)
processauditpolicy : P(USER × FILENAME × ProcessAction)
log : seq LogElement

∀p1, p2 : processlist • p1.process_id = p2.process_id ↔ p1 = p2
∀p : processlist • p.filename ∈ dom filelist ∧ p.owner ∈ userlist

InitSYSTEM

system : SYSTEM

#system.filelist = 1
#system.userlist = 1
#system.processlist = 1
system.log = ∅

AddUser_OK

ASM SYSTEM

newUser? : USER
userlogelement : UserLogElement
logelement : LogElement
r! : Response

newUser? ∈ userlist
userlist′ = userlist ∪ {newUser?}
r! = OK
(newUser?, Add_User) ∈ userauditpolicy ∨ (userlogelement.user = newUser? ∧
userlogelement.action = Add_User ∧ logelement.r = r! ∧
logelement.u = {userlogelement} ∧ logelement.f = ∅ ∧
logelement.p = ∅ ∧ log′ = log \append{logelement})
AddUser_Exists

\[\text{ASYSTEM}\]
\[\text{newUser? : USER}\]
\[\text{userlogelement : UserLogElement}\]
\[\text{logelement : LogElement}\]
\[r! : \text{Response}\]
\[\text{newUser? } \in \text{userlist}\]
\[r! = \text{UserExists}\]
\[(\text{newUser?}, \text{Add_User}) \in \text{userauditpolicy} \lor (\text{userlogelement.user} = \text{newUser?} \land \text{userlogelement.action} = \text{Add_User} \land \text{logelement.r} = r! \land \text{logelement.u} = \{\text{userlogelement}\} \land \text{logelement.f} = \emptyset \land \text{logelement.p} = \emptyset \land \text{log'} = \text{log}^{-1}(\text{logelement})]\]

AddUser \equiv AddUser_OK \lor AddUser_Exists

DeleteUser_OK

\[\text{ASYSTEM}\]
\[\text{user? : USER}\]
\[\text{userlogelement : UserLogElement}\]
\[\text{logelement : LogElement}\]
\[r! : \text{Response}\]
\[\text{user? } \in \text{userlist}\]
\[\forall p : \text{processlist} \bullet p.\text{owner} \neq \text{user?}\]
\[\text{userlist'} = \text{userlist} \setminus \{\text{user?}\}\]
\[r! = \text{OK}\]
\[(\text{user?}, \text{Delete_User}) \in \text{userauditpolicy} \lor (\text{userlogelement.user} = \text{user?} \land \text{userlogelement.action} = \text{Delete_User} \land \text{logelement.r} = r! \land \text{logelement.u} = \{\text{userlogelement}\} \land \text{logelement.f} = \emptyset \land \text{logelement.p} = \emptyset \land \text{log'} = \text{log}^{-1}(\text{logelement})]\]
__DeleteUser_NotExists__

\[
\text{\Delta SYSTEM}
\]

\[
\text{user? : USER}
\]

\[
\text{userlogelement : UserLogElement}
\]

\[
\text{logelement : LogElement}
\]

\[
\text{r! : Response}
\]

\[
\text{user? \in userlist}
\]

\[
\text{r! = UserNotExists}
\]

\[
(\text{user?, Delete_User}) \in userauditpolicy \lor (\text{userlogelement.user = user?} \land \\
\text{userlogelement.action = Delete_User} \land \text{logelement.r = r!} \land \\
\text{logelement.u = \{userlogelement\} \land logelement.f = \emptyset} \land \\
\text{logelement.p = \emptyset \land log' = log^<(logelement)\})
\]

__DeleteUser_RunningProcess__

\[
\text{\Delta SYSTEM}
\]

\[
\text{user? : USER}
\]

\[
\text{userlogelement : UserLogElement}
\]

\[
\text{logelement : LogElement}
\]

\[
\text{r! : Response}
\]

\[
\text{user? \in userlist}
\]

\[
\exists p : processlist : p.owner = user?
\]

\[
(\text{user?, Delete_User}) \in userauditpolicy \lor (\text{userlogelement.user = user?} \land \\
\text{userlogelement.action = Delete_User} \land \text{logelement.r = r!} \land \\
\text{logelement.u = \{userlogelement\} \land logelement.f = \emptyset} \land \\
\text{logelement.p = \emptyset \land log' = log^<(logelement)\})
\]

\[
\text{r! = UserRunningProcess}
\]

\[
\text{DeleteUser} \equiv \text{DeleteUser_OK} \lor \text{DeleteUser_NotExists} \lor \text{DeleteUser_RunningProcess}
\]
AddFile_OK

ΔSYSTEM
filename?: FILENAME
fileproperties?: FILE_PROPERTIES
filelogelement: FileLogElement
logelement: LogElement
r!: Response

filename? ∈ dom filelist
filelist' = filelist ∪ \{filename? → fileproperties?\}
r! = OK
(filename?, Add_File) ∈ fileauditpolicy ∨ (filelogelement.filename= filename? ∧
filelogelement.action = Add_File ∧ logelement.r = r! ∧
logelement.u = ∅ ∧ logelement.f = {filelogelement} ∧
logelement.p = ∅ ∧ log' = log~(logelement))

AddFile_Exists

ΔSYSTEM
filename?: FILENAME
fileproperties?: FILE_PROPERTIES
filelogelement: FileLogElement
logelement: LogElement
r!: Response

filename? ∈ dom filelist
r! = FileExists
(filename?, Add_File) ∈ fileauditpolicy ∨ (filelogelement.filename= filename? ∧
filelogelement.action = Add_File ∧ logelement.r = r! ∧
logelement.u = ∅ ∧ logelement.f = {filelogelement} ∧
logelement.p = ∅ ∧ log' = log~(logelement))

AddFile = AddFile_OK ∨ AddFile_Exists
\_ReadFile\_OK\_

\textit{SYSTEM}

\texttt{filename? : FILENAME}
\texttt{fileproperties! : FILE\_PROPERTIES}
\texttt{filelogelement : FileLogElement}
\texttt{logelement : LogElement}
\texttt{r! : Response}

\texttt{filename? \in dom filelist}
\texttt{fileproperties! = filelist(filename?)}
\texttt{r! = OK}
\texttt{(filename?, Read\_File) \in fileauditpolicy \lor (filelogelement.filename = filename? \land}
\texttt{filelogelement.action = Read\_File \land logelement.r = r! \land}
\texttt{logelement.u = \emptyset \land logelement.f = \{filelogelement\} \land}
\texttt{logelement.p = \emptyset \land log' = log'(logelement))}

\_ReadFile\_NotExists\_

\textit{SYSTEM}

\texttt{filename? : FILENAME}
\texttt{fileproperties! : FILE\_PROPERTIES}
\texttt{filelogelement : FileLogElement}
\texttt{logelement : LogElement}
\texttt{r! : Response}

\texttt{filename? \in dom filelist}
\texttt{r! = FileNotExists}
\texttt{(filename?, Read\_File) \in fileauditpolicy \lor (filelogelement.filename = filename? \land}
\texttt{filelogelement.action = Read\_File \land logelement.r = r! \land}
\texttt{logelement.u = \emptyset \land logelement.f = \{filelogelement\} \land}
\texttt{logelement.p = \emptyset \land log' = log'(logelement))}

\texttt{ReadFile \equiv ReadFile\_OK \lor ReadFile\_NotExists}
DeleteFile_OK

\[\text{SYSTEM}\]
filename?: FILENAME
filelogelement: FileLogElement
logelement: LogElement
r!: Response

filename? \in \text{dom filelist}
filelist' = filelist \setminus \{filename?\} \rightarrow \text{filelist}(filename?)
r! = OK

(filename?, Delete_File) \in \text{fileauditpolicy} \land (filelogelement.filename = filename? \land
filelogelement.action = Delete_File \land logelement.r = r! \land
logelement.u = \emptyset \land logelement.f = \{filelogelement\} \land
logelement.p = \emptyset \land log' = log'(logelement))

DeleteFile_NotExists

\[\text{SYSTEM}\]
filename?: FILENAME
filelogelement: FileLogElement
logelement: LogElement
r!: Response

filename? \in \text{dom filelist}
r! = FileNotExists

(filename?, Delete_File) \in \text{fileauditpolicy} \lor (filelogelement.filename = filename? \land
filelogelement.action = Delete_File \land logelement.r = r! \land
logelement.u = \emptyset \land logelement.f = \{filelogelement\} \land
logelement.p = \emptyset \land log' = log'(logelement))
\textbf{DeleteFile\_FileBeingExecuted}

\[\Delta \text{SYSTEM}
\]
\text{filename? : FILENAME}
\text{filelogelement : FileLogElement}
\text{logelement : LogElement}
\text{r! : Response}

\text{filename? \in dom\ filelist}
\exists p : processlist \bullet p.\ filename = filename?
\text{r! = FileBeingExecuted}

\text{(filename?, Delete\_File) \in fileauditpolicy \lor (filelogelement.\ filename = filename? \land}
\text{filelogelement.\ action = Delete\_File \land logelement.\ r = r! \land}
\text{logelement.\ u = \emptyset \land logelement.\ f = \{filelogelement\} \land}
\text{logelement.\ p = \emptyset \land log' = log^\sim(logelement))}

\text{DeleteFile \equiv DeleteFile\_OK \lor DeleteFile\_NotExists \lor}
\text{DeleteFile\_FileBeingExecuted}
\_ChangeFile\_OK\_
\Delta\text{SYSTEM}\\
\text{filename}? : \text{FILENAME}\\
\text{properties}? : \text{FILE\_PROPERTIES}\\
\text{filechange} : \text{FILE\_CHANGE}\\
\text{filelogelement} : \text{FileLogElement}\\
\text{logelement}! : \text{LogElement}\\
\text{r! : Response}\\
\text{filename}? \in \text{dom filelist}\\
\text{filelist}' = \text{filelist} \oplus \{\text{filename}? \mapsto \text{properties}??\}\\
\text{r!} = \text{OK}\\
(\text{filename}?, \text{Change\_File}) \in \text{fileauditpolicy} \lor (\text{filelogelement}.\text{filename} = \text{filename}? \land \\
\text{filelogelement}.\text{action} = \text{Change\_File} \land \\
\text{filechange} = \text{FILE\_PROPERTIES\_MAP(}\text{filelist(}\text{filename}?), \text{properties}??\}) \land \\
\text{FILE\_CHANGE\_MAP(}\text{filelist(}\text{filename}?), \text{filechange}) = \text{properties}? \land \\
\text{filelogelement}.\text{filechange} = \text{filechange} \land \text{logelement}!.r = \text{r!} \land \\
\text{logelement}!.u = \emptyset \land \text{logelement}!.f = \{\text{filelogelement}\} \land \\
\text{logelement}!.p = \emptyset)

\_ChangeFile\_NotExists\_
\Delta\text{SYSTEM}\\
\text{filename}? : \text{FILENAME}\\
\text{properties}? : \text{FILE\_PROPERTIES}\\
\text{filelogelement} : \text{FileLogElement}\\
\text{logelement} : \text{LogElement}\\
\text{r! : Response}\\
\text{filename}? \in \text{dom filelist}\\
\text{r!} = \text{FileNotFoundException}\\
(\text{filename}?, \text{Change\_File}) \in \text{fileauditpolicy} \lor (\text{filelogelement}.\text{filename} = \text{filename}? \land \\
\text{filelogelement}.\text{action} = \text{Change\_File} \land \text{logelement}.r = \text{r!} \land \\
\text{logelement}.u = \emptyset \land \text{logelement}.f = \{\text{filelogelement}\} \land \\
\text{logelement}.p = \emptyset \land \text{log'} = \text{log}^{-1}(\text{logelement}))

\text{ChangeFile} \equiv \text{ChangeFile\_OK} \lor \text{ChangeFile\_NotExists}
\textit{StartProcess\_OK}

\textbf{ΔSYSTEM}

\textbf{process? : PROCESS}

\textbf{processlogelement : ProcessLogElement}

\textbf{logelement : LogElement}

\textbf{r! : Response}

\textbf{process?.owner ∈ userlist}

\textbf{process?.filename ∈ dom filelist}

\(∀p:processlist \bullet p\.process\_id \neq process?.process\_id\)

\textbf{processlist' = processlist \cup \{process?\}}

\textbf{r! = OK}

\(\langle process?.owner, process?.filename, \text{Start\_Process} \rangle ∈ \text{processauditpolicy} \lor\)

\(\langle processlogelement\.process\_id = process?.process\_id \land\)

\textbf{processlogelement.filename = process?.filename \land}

\textbf{processlogelement.owner = process?.owner \land processlogelement.action = \text{Start\_Process} \land}

\textbf{logelement.r = r! \land logelement.u = \emptyset \land logelement.f = \emptyset \land}

\textbf{logelement.p = \{processlogelement\} \land log' = log'\{logelement\})}

\textit{StartProcess\_FileNotExists}

\textbf{ΔSYSTEM}

\textbf{process? : PROCESS}

\textbf{processlogelement : ProcessLogElement}

\textbf{logelement : LogElement}

\textbf{r! : Response}

\textbf{process?.filename ∈ dom filelist}

\textbf{r! = FileNotExists}

\(\langle process?.owner, process?.filename, \text{Start\_Process} \rangle ∈ \text{processauditpolicy} \lor\)

\(\langle processlogelement\.process\_id = process?.process\_id \land\)

\textbf{processlogelement.filename = process?.filename \land}

\textbf{processlogelement.owner = process?.owner \land processlogelement.action = \text{Start\_Process} \land}

\textbf{logelement.r = r! \land logelement.u = \emptyset \land logelement.f = \emptyset \land}

\textbf{logelement.p = \{processlogelement\} \land log' = log'\{logelement\})}
\[ \text{StartProcess} \equiv \text{StartProcess_OK} \lor \text{StartProcess_FileNotExists} \lor \text{StartProcess_UserNotExists} \]
StopProcess_OK

\[\Delta \text{SYSTEM} \]

\[\text{process\_id}\? : \text{PROCESS\_ID} \]
\[\text{processlogoelement} : \text{ProcessLogElement} \]
\[\text{process} : \text{PROCESS} \]
\[\text{logelement} : \text{LogElement} \]
\[r! : \text{Response} \]

\[\exists p:\text{processlist} \bullet p.\text{process\_id} = \text{process\_id}\? \]
\[\text{processlist}' = \text{processlist}\{p.\text{PROCESS}|p.\text{process\_id} = \text{process\_id}\?\} \]
\[r! = \text{OK} \]

\[\exists p_1 : \text{processlist} \bullet \text{process} = p_1 \land p_1.\text{process\_id} = \text{process\_id}\? \]
\[(\text{process.}\text{owner}, \text{process.}\text{filename}, \text{Stop\_Process}) \in \text{processauditpolicy} \land \]
\[(\text{processlogoelement.}\text{process\_id} = \text{process}\text{.process\_id} \land \]
\[\text{processlogoelement.}\text{filename} = \text{process}\text{.filename} \land \]
\[\text{processlogoelement.}\text{owner} = \text{process}\text{.owner} \land \text{processlogoelement.}\text{action} = \text{Stop\_Process} \land \]
\[\text{logelement.r} = r! \land \text{logelement.u} = \emptyset \land \text{logelement.f} = \emptyset \land \]
\[\text{logelement.p} = \{\text{processlogoelement}\} \land \log' = \log^+(\text{logelement}) \]

StopProcess_ProcessIDNotExists

\[\equiv \text{SYSTEM} \]

\[\text{process\_id}\? : \text{PROCESS\_ID} \]
\[\text{processlogoelement} : \text{ProcessLogElement} \]
\[\text{logelement} : \text{LogElement} \]
\[r! : \text{Response} \]

\[\forall p:\text{processlist} \bullet p.\text{process\_id} \neq \text{process\_id}\? \]
\[r! = \text{ProcessIDNotExists} \]

\[(\forall \text{owner} : \text{USER} \bullet (\forall \text{filename} : \text{FILENAME} \bullet \]
\[\quad (\text{owner, filename, Stop\_Process}) \in \text{processauditpolicy}) \land \]
\[(\text{processlogoelement.}\text{action} = \text{Stop\_Process} \land \]
\[\text{logelement.r} = r! \land \text{logelement.u} = \emptyset \land \text{logelement.f} = \emptyset \land \]
\[\text{logelement.p} = \{\text{processlogoelement}\} \land \log' = \log^+(\text{logelement}) \]
StopProcess \equiv \text{StopProcess\_OK} \lor \text{StopProcess\_ProcessIDNotExists}

\text{ChangeProcess\_OK}

\begin{align*}
\Delta \text{SYSTEM}\\
\text{process\_id}? &: \text{PROCESS\_ID}\\
\text{newstate}? &: \text{PROCESS\_STATE}\\
\text{processchange} &: \text{PROCESS\_STATE\_CHANGE}\\
\text{processlogelement} &: \text{ProcessLogElement}\\
\text{logelement} &: \text{LogElement}\\
\text{r!} &: \text{Response}\\
\end{align*}

\begin{align*}
\text{(let } p = \{ p1: \text{processlist} \mid p1.\text{process\_id} = \text{process\_id}? \} &\cdot\\
(\forall p2: p \bullet p2.\text{state} = \text{newstate}?) &\land \#p = 1 \land\\
\text{processlist'} = \text{processlist}\setminus\{ p2: \text{processlist} \mid p2.\text{process\_id} = \text{process\_id}? \} &\land\\
\text{processlist'} = \text{processlist'} \cup p &\land\\
\text{r!} = \text{OK} &\land\\
(\forall p1: p \bullet (p1.\text{owner}, p1.\text{filename}, \text{Change\_Process}) \in \text{processauditpolicy} &\lor\\
\text{processlogelement.\text{process\_id}} = \text{process\_id}? &\land\\
\text{processlogelement.\text{filename}} = p1.\text{filename} &\land\\
\text{processlogelement.\text{owner}} = p1.\text{owner} &\land\\
\text{processlogelement.\text{action}} = \text{Change\_Process} &\land\\
\text{processchange} = \text{PROCESS\_STATE\_MAP}(p1.\text{state}, \text{newstate}?) &\land\\
\text{newstate}? = \text{PROCESS\_STATE\_CHANGE\_MAP}(p1.\text{state}, \text{processchange}) &\land\\
\text{processlogelement.\text{processchange}} = \text{processchange} &\land\\
\text{logelement.r} = \text{r!} &\land \text{logelement.u} = \emptyset &\land \text{logelement.f} = \emptyset &\land\\
\text{logelement.p} = (\text{processlogelement}) &\land \log' = \log''(\text{logelement})))))
\end{align*}
\_ChangeProcess\_ProcessIDNotExists

\[ \exists SYSTEM
\]

process\_id?: PROCESS\_ID
newstate?: PROCESS\_STATE
processlogelement: ProcessLogElement
logelement: LogElement
r\!: Response

\[ \forall p:processlist \bullet p.process\_id \neq process\_id? \]
\[ r\! = \text{ProcessIDNotExists} \]
\[ (\forall owner: USER \bullet (\forall filename: FILENAME \bullet
\hspace{1cm} \text{(owner, filename, Change\_Process) } \notin \text{processauditpolicy}) \vee
\hspace{1cm} \text{(processlogelement.action = Change\_Process \land
\hspace{1cm} logelement.r = r\! \land logelement.u = \varnothing \land logelement.f = \varnothing \land
\hspace{1cm} logelement.p = \{processlogelement\} \land log' = log:\{logelement\})} \]

\[ \text{ChangeProcess} \equiv \text{ChangeProcess\_OK} \lor \text{ChangeProcess\_ProcessIDNotExists} \]
APPENDIX G

Intrusion Detection Tactic System Model Specification

[USER]
[FILENAME]
[FILE_PROPERTIES]
[FILE_CHANGE]
[PROCESS_STATE]
[PROCESS_STATE_CHANGE]
[PROCESS_ID]

UserAction ::= Add_User|Delete_User
FileAction ::= Add_File|Delete_File|Read_File|Change_File
ProcessAction ::= Start_Process|Stop_Process|Change_Process
Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|
ProcessIDNotExists|UserRunningProcess|FileBeingExecuted

FILE_CHANGE_MAP : FILE_PROPERTIES × FILE_CHANGE
⇒ FILE_PROPERTIES

FILE_PROPERTIES_MAP : FILE_PROPERTIES × FILE_PROPERTIES
⇒ FILE_CHANGE

PROCESS_STATE_CHANGE_MAP : PROCESS_STATE × PROCESS_STATE_CHANGE
⇒ PROCESS_STATE

PROCESS_STATE_MAP : PROCESS_STATE × PROCESS_STATE
⇒ PROCESS_STATE_CHANGE
_UserLogElement_
user: USER
action : UserAction

_FileLogElement_
filename : FILENAME
action : FileAction
filechange : FILE_CHANGE

_ProcessLogElement_
process_id : PROCESS_ID
filename : FILENAME
owner: USER
action : ProcessAction
processchange : PROCESS_STATE_CHANGE

_LogElement_
u : PUserLogElement
f : PFileLogElement
p : PProcessLogElement
r : Response
#u + #f + #p = 1
IDS_ALERT

\[ \subseteq \text{SYSTEM} \]
\[ \text{sublog} : \mathbb{P} (P \cap \cap) \]
\[ \text{newalertlog} : \text{seq} (\text{seq}, \text{LogElement}) \]

\[ \text{log}' \neq \text{log} \]
\[ \forall s_l : \text{sublog} \bullet s_l \text{in} \text{idspolicy \land } (s_l \text{in} \text{newalertlog} \land ) \]
\[ (\exists p : \mathbb{N}_0 \bullet (\text{lastIDSAnalysis} < p \leq \#\text{log}' \land p \in s_l)) \]
\[ \text{idsalertlist}' = \text{idsalertlist}' \text{newalertlog} \]
\[ \text{lastIDSAnalysis}' = \#\text{log}' \]

IDS_NO_ALERT

\[ \subseteq \text{SYSTEM} \]
\[ \text{log}' = \text{log} \]
\[ \text{idsalertlist}' = \text{idsalertlist} \]

IDS_System \equiv IDS_ALERT \lor IDS_NO_ALERT

PROCESS

filename : FILENAME
owner : USER
process_id : PROCESS_ID
state : PROCESS_STATE
```plaintext
_SYSTEM_

filelist : FILENAME→FILE_PROPERTIES
userlist : P USER
processlist : P PROCESS
userlogpolicy : P(USER × UserAction)
filelogpolicy : P(FILENAME × FileAction)
processlogpolicy : P(USER × FILENAME × ProcessAction)
log : seq LogElement
idspolicy : P(seq, LogElement)
idsalertlist: seq(seq, LogElement)
lastIDSAnalysis : ℕ

∀p1, p2 : processlist • p1.process_id = p2.process_id ↔ p1 = p2
∀p : processlist • p.filename ∈ dom filelist ∧ p.owner ∈ userlist

_InitSYSTEM_

system : SYSTEM

#system.filelist = 1
#system.userlist = 1
#system.processlist = 1
system.log = ∅
system.idsalertlist = ∅
system.lastIDSAnalysis = 0
```
AddUser_OK

ASYSTEM
IDS_System
newUser? : USER
userlogelement : UserLogElement
logelement : LogElement
r! : Response

newUser? ∈ userlist
userlist' = userlist ∪ {newUser?}
r! = OK
(newUser?, Add_User) ∈ userlogpolicy ∨ (userlogelement.user = newUser? ∧
userlogelement.action = Add_User ∧ logelement.r = r! ∧
logelement.u = {userlogelement} ∧ logelement.f = ∅ ∧
logelement.p = ∅ ∧ log' = log^"(logelement)"

AddUser_Exists

ASYSTEM
IDS_System
newUser? : USER
userlogelement : UserLogElement
logelement : LogElement
r! : Response

newUser? ∈ userlist
r! = UserExists
(newUser?, Add_User) ∈ userlogpolicy ∨ (userlogelement.user = newUser? ∧
userlogelement.action = Add_User ∧ logelement.r = r! ∧
logelement.u = {userlogelement} ∧ logelement.f = ∅ ∧
logelement.p = ∅ ∧ log' = log^"(logelement)"

AddUser ≡ AddUser_OK ∨ AddUser_Exists
\_\_DeleteUser\_\_OK\_
\__\ASYSTEM
\_\_IDS\_System
user? : USER
userlogelement : UserLogElement
logelement : LogElement
r! : Response
user? \in userlist
\forall p : processlist \cdot p.owner \neq user?
userlist' = userlist \setminus \{user?\}
r! = OK
(user?, Delete_User) \in userlogpolicy \lor (userlogelement.user = user? \land
userlogelement.action = Delete_User \land logelement.r = r! \land
logelement.u = \{userlogelement\} \land logelement.f = \emptyset \land
logelement.p = \emptyset \land log' = log \setminus \{logelement\})

\_\_DeleteUser\_\_NotExists\_
\__\ASYSTEM
\_\_IDS\_System
user? : USER
userlogelement : UserLogElement
logelement : LogElement
r! : Response
user? \notin userlist
r! = UserNotExists
(user?, Delete_User) \in userlogpolicy \lor (userlogelement.user = user? \land
userlogelement.action = Delete_User \land logelement.r = r! \land
logelement.u = \{userlogelement\} \land logelement.f = \emptyset \land
logelement.p = \emptyset \land log' = log \setminus \{logelement\})
**DeleteUser** RunningProcess

\[ \text{DeleteUser} \equiv \text{DeleteUser} \_\text{OK} \lor \text{DeleteUser} \_\text{NotExists} \lor \]

\[ \text{DeleteUser} \_\text{RunningProcess} \]

\[ \]
AddFile _ Exists
\newcommand{\System}{\textsc{asystem}}
\newcommand{\idsystem}{\textsc{ids} \_ \textsc{system}}
\newcommand{\filename}{\texttt{filename}}
\newcommand{\fileproperties}{\texttt{fileproperties}}
\newcommand{\filelogelement}{\texttt{filelogelement}}
\newcommand{\logelement}{\texttt{logelement}}
\newcommand{\r}{\texttt{r}}
\newcommand{\addfile}{addfile}
\newcommand{\fileexists}{fileexists}
\newcommand{\filelogpolicy}{filelogpolicy}
\newcommand{\filelist}{filelist}
\newcommand{\addfileok}{addfileok}
\newcommand{\log}{\texttt{log}}

\begin{itemize}
  \item \filename \in \text{dom filelist}
  \item \r = \text{FileExists}
  \item (\filename, \addfile) \in \filelogpolicy \lor (\filelogelement.\filename = \filename \land \filelogelement.\action = \addfile \land \logelement.\r = \r \land \logelement.\u = \emptyset \land \logelement.\f = \{\filelogelement\} \land \logelement.\p = \emptyset \lor \log' = \log' (\logelement))
\end{itemize}

AddFile \equiv AddFile _ OK \lor AddFile _ Exists

AddFile _ OK
\newcommand{\addfileok}{addfileok}
\newcommand{\fileproperties}{\texttt{fileproperties}}
\newcommand{\filename}{\texttt{filename}}
\newcommand{\addfile}{addfile}
\newcommand{\filelogpolicy}{filelogpolicy}
\newcommand{\filelist}{filelist}
\newcommand{\addfileok}{addfileok}
\newcommand{\log}{\texttt{log}}

\begin{itemize}
  \item \filename \in \text{dom filelist}
  \item \logelement = \text{filelist}(\filename)
  \item \r = \text{OK}
  \item (\filename, \addfile) \in \filelogpolicy \lor (\filelogelement.\filename = \filename \land \filelogelement.\action = \addfile \land \logelement.\r = \r \land \logelement.\u = \emptyset \land \logelement.\f = \{\filelogelement\} \land \logelement.\p = \emptyset \lor \log' = \log' (\logelement))
\end{itemize}
ReadFile_NotExists

\[\Delta \text{SYSTEM}\]
\[\Delta \text{IDS System}\]
filename? : FILENAME
fileproperties! : FILE_PROPERTIES
filelogelement : FileLogElement
logelement : LogElement
r! : Response

filename? \in dom filelist
r! = FileNotExists

(filename?, Read_File) \in filelogpolicy \lor (filelogelement.filename = filename? \land
filelogelement.action = Read_File \land logelement.r = r! \land
logelement.u = \emptyset \land logelement.f = \{filelogelement\} \land
logelement.p = \emptyset \land log^' = log^\{logelement\})

ReadFile \equiv ReadFile_OK \lor ReadFile_NotExists

DeleteFile_OK

\[\Delta \text{SYSTEM}\]
\[\Delta \text{IDS System}\]
filename? : FILENAME
filelogelement : FileLogElement
logelement : LogElement
r! : Response

filename? \in dom filelist
\forall p : \text{processlist} \bullet p.filename \neq filename?
filelist' = filelist \setminus \{filename?\rightarrow filelist(filename?)\}

r! = OK

(filename?, Delete_File) \in filelogpolicy \lor (filelogelement.filename = filename? \land
filelogelement.action = Delete_File \land logelement.r = r! \land
logelement.u = \emptyset \land logelement.f = \{filelogelement\} \land
logelement.p = \emptyset \land log^' = log^\{logelement\})
DeleteFile_NotExists

\[\Delta\text{SYSTEM}\]
\[\text{IDS\_System}\]
\[\text{filename}\? : \text{FILENAME}\]
\[\text{filelogelement : FileLogElement}\]
\[\text{logelement : LogElement}\]
\[\text{r! : Response}\]

\[\text{filename}\? \in \text{dom\_filelist}\]
\[\text{r!} = \text{FileNotExists}\]
\[(\text{filename}\?, \text{Delete\_File}) \in \text{filelogpolicy} \lor (\text{filelogelement. filename} = \text{filename}\? \land \text{filelogelement.action} = \text{Delete\_File}) \land \text{logelement.r} = \text{r!} \land \text{logelement.u} = \emptyset \land \text{logelement.f} = \{\text{filelogelement}\} \land \text{logelement.p} = \emptyset \land \text{log'} = \text{log}\sim(\text{logelement})\]

DeleteFile_FileBeingExecuted

\[\Delta\text{SYSTEM}\]
\[\text{IDS\_System}\]
\[\text{filename}\? : \text{FILENAME}\]
\[\text{filelogelement : FileLogElement}\]
\[\text{logelement : LogElement}\]
\[\text{r! : Response}\]

\[\text{filename}\? \in \text{dom\_filelist}\]
\[\exists p : \text{processlist} . p.\text{filename} = \text{filename}\?\]
\[\text{r!} = \text{FileBeingExecuted}\]
\[(\text{filename}\?, \text{Delete\_File}) \in \text{filelogpolicy} \lor (\text{filelogelement. filename} = \text{filename}\? \land \text{filelogelement.action} = \text{Delete\_File}) \land \text{logelement.r} = \text{r!} \land \text{logelement.u} = \emptyset \land \text{logelement.f} = \{\text{filelogelement}\} \land \text{logelement.p} = \emptyset \land \text{log'} = \text{log}\sim(\text{logelement})\]

\[\text{DeleteFile} \equiv \text{DeleteFile\_OK} \lor \text{DeleteFile\_NotExists} \lor \text{DeleteFile\_FileBeingExecuted}\]
ChangeFile_OK

\[\Delta\text{SYSTEM}\]
\[\Delta\text{IDS\_System}\]
filename?: FILENAME
properties?: FILE\_PROPERTIES
filechange: FILE\_CHANGE
filelogelement: FileLogElement
logelement!: LogElement
r!: Response

filename? \in \text{dom\ filelist}
filelist' = filelist \oplus \{filename? \mapsto properties?\}
r! = OK
(filename?, \text{Change\_File}) \notin \text{filelogpolicy} \lor (filelogelement.\text{filename} = filename? \land
filelogelement.\text{action} = \text{Change\_File} \land
filechange = FILE\_PROPERTIES\_MAP(filelist(filename?), properties?) \land
FILE\_CHANGE\_MAP(filelist(filename?), filechange) = properties? \land
filelogelement.\text{filechange} = filechange \land logelement!.r = r! \land
logelement!.u = \emptyset \land logelement!.f = \{filelogelement\} \land
logelement!.p = \emptyset)

ChangeFile_NotExists

\[\Delta\text{SYSTEM}\]
\[\Delta\text{IDS\_System}\]
filename?: FILENAME
properties?: FILE\_PROPERTIES
filelogelement: FileLogElement
logelement: LogElement
r!: Response

filename? \notin \text{dom\ filelist}
r! = \text{FileNotExists}
(filename?, \text{Change\_File}) \notin \text{filelogpolicy} \lor (filelogelement.\text{filename} = filename? \land
filelogelement.\text{action} = \text{Change\_File} \land logelement!.r = r! \land
logelement!.u = \emptyset \land logelement!.f = \{filelogelement\} \land
logelement!.p = \emptyset \land \log' = \log^{-1}(logelement)
\[ ChangeFile \equiv ChangeFile_{OK} \lor ChangeFile_{NotExists} \]

\[ StartProcess_{OK} \]
\[ \DeltaSYSTEM \]
\[ IDS\_System \]
\[ process? : PROCESS \]
\[ processlogelement : ProcessLogElement \]
\[ logelement : LogElement \]
\[ r! : Response \]

\[ process?.owner \in userlist \]
\[ process?.filename \in dom\_filelist \]
\[ \forall p:processlist \bullet p.process\_id \neq process?.process\_id \]
\[ processlist' = processlist \cup \{process?\} \]
\[ r! = OK \]
\[ (process?.owner, process?.filename, Start\_Process) \in processlogpolicy \lor \]
\[ (processlogelement.process\_id = process?.process\_id \land \]
\[ processlogelement.filename = process?.filename \land \]
\[ processlogelement.owner = process?.owner \land \]
\[ processlogelement.action = Start\_Process \land \]
\[ logelement.r = r! \land logelement.u = \emptyset \land logelement.f = \emptyset \land \]
\[ logelement.p = \{processlogelement\} \land log' = \text{log}^{-1}(logelement) \]
\_StartProcess\_FileNotExists
\begin{align*}
\Delta & \text{SYSTEM} \\
& \text{IDS\_System} \\
& \text{process? : PROCESS} \\
& \text{processlogelement : ProcessLogElement} \\
& \text{logelement : LogElement} \\
& r! : \text{Response} \\
& \text{process?.filename} \notin \text{dom\_filelist} \\
& r! = \text{FileNotExists} \\
& (\text{process?.owner, process?.filename,Start\_Process}) \notin \text{processlogpolicy} \lor \\
& (\text{processlogelement.process\_id = process?.process\_id} \land \\
& \text{processlogelement.filename = process?.filename} \land \\
& \text{processlogelement.owner = process?.owner} \land \\
& \text{processlogelement.action = Start\_Process} \land \\
& \text{logelement.r = r} \land \text{logelement.u = } \emptyset \land \text{logelement.f = } \emptyset \land \\
& \text{logelement.p = \{processlogelement\} \land log' = log^{-1}\{logelement\}}
\end{align*}

\_StartProcess\_UserNotExists
\begin{align*}
\Delta & \text{SYSTEM} \\
& \text{IDS\_System} \\
& \text{process? : PROCESS} \\
& \text{processlogelement : ProcessLogElement} \\
& \text{logelement : LogElement} \\
& r! : \text{Response} \\
& \text{process?.owner} \notin \text{userlist} \\
& r! = \text{UserNotExists} \\
& (\text{process?.owner, process?.filename,Start\_Process}) \notin \text{processlogpolicy} \lor \\
& (\text{processlogelement.process\_id = process?.process\_id} \land \\
& \text{processlogelement.filename = process?.filename} \land \\
& \text{processlogelement.owner = process?.owner} \land \\
& \text{processlogelement.action = Start\_Process} \land \\
& \text{logelement.r = r} \land \text{logelement.u = } \emptyset \land \text{logelement.f = } \emptyset \land \\
& \text{logelement.p = \{processlogelement\} \land log' = log^{-1}\{logelement\}}
\end{align*}
\[ \text{StartProcess} \equiv \text{StartProcess\_OK} \lor \text{StartProcess\_FileNotExists} \lor \text{StartProcess\_UserNotExists} \]

\[ \Delta \text{SYSTEM} \]
\[ \text{IDS\_System} \]
\[ \text{process:\_id?} : \text{PROCESS\_ID} \]
\[ \text{processlogelement} : \text{ProcessLogElement} \]
\[ \text{process} : \text{PROCESS} \]
\[ \text{logelement} : \text{LogElement} \]
\[ r! : \text{Response} \]

\[ \exists p : \text{processlist} : \bullet p.\text{process\_id} = \text{process\_id}? \]
\[ \text{processlist'} = \text{processlist}\{p:\text{PROCESS}|p.\text{process\_id} = \text{process\_id}?\} \]
\[ r! = \text{OK} \]

\[ \exists p_1 : \text{processlist} : \bullet \text{process} = p_1 \land p_1.\text{process\_id} = \text{process\_id}? \]
\[ (\text{process.\_owner}, \text{process.\_filename}, \text{Stop\_Process}) \in \text{processlogpolicy} \lor \]
\[ (\text{processlogelement.\_process\_id} = \text{process.\_process\_id} \land \]
\[ \text{processlogelement.\_filename} = \text{process.\_filename} \land \]
\[ \text{processlogelement.\_owner} = \text{process.\_owner} \land \]
\[ \text{processlogelement.\_action} = \text{Stop\_Process} \land \]
\[ \text{logelement.\_r} = r! \land \text{logelement.\_u} = \emptyset \land \text{logelement.\_f} = \emptyset \land \]
\[ \text{logelement.\_p} = \{\text{processlogelement}\} \land \log' = \log\{\text{logelement}\} \]
_StopProcess_ProcessIDNotExists_

ASYSTEM
IDS_System

process_id? : PROCESS_ID
processlogelement : ProcessLogElement
process : PROCESS
logelement : LogElement
r! : Response

\forall p:processlist \bullet p.process_id \neq process_id?

r! = ProcessIDNotExists

(\forall owner: USER \bullet (\forall filename: FILENAME \bullet

( owner, filename, Stop_Process) \notin processlogpolicy)) \lor

(processlogelement.action = Stop_Process \land

logelement.r = r! \land logelement.u = \emptyset \land logelement.f = \emptyset \land

logelement.p = \{processlogelement\} \land \log' = \log^-(logelement))

StopProcess \equiv StopProcess_OK \lor StopProcess_ProcessIDNotExists
_ChangeProcess_OK_

ΔSYSTEM

IDS_System

process_id?: PROCESS_ID

newstate?: PROCESS_STATE

processchange: PROCESS_STATE_CHANGE

processlogelement: ProcessLogElement

logelement: LogElement

r!: Response

(let p = {p1: processlist | p1.process_id = process_id?} •

(∀p2:p • p2.state = newstate?) ∧ #p = 1 ∧

processlist' = processlist\{p2: processlist | p2.process_id = process_id?} ∧

processlist' = processlist' ∪ p ∧

r! = OK ∧

(∀p1:p • (p1.owner, p1.filename, Change_Process) ∈ processlogpolicy ∨

(processlogelement.process_id = process_id? ∧

processlogelement.filename = p1.filename ∧

processlogelement.owner = p1.owner ∧

processlogelement.action = Change_Process ∧

processchange = PROCESS_STATE_MAP(p1.state, newstate?) ∧

newstate? = PROCESS_STATE_CHANGE_MAP(p1.state, processchange) ∧

processlogelement.processchange = processchange ∧

logelement.r = r! ∧ logelement.u = ∅ ∧ logelement.f = ∅ ∧

logelement.p = {processlogelement} ∧ log' = log'⟨logelement⟩))))
\textbf{\textit{ChangeProcess\_ProcessIDNotExists}}

<table>
<thead>
<tr>
<th>\textsl{SYSTEM}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsl{IDS_System}</td>
</tr>
<tr>
<td>\textit{process_id}? : \textit{PROCESS_ID}</td>
</tr>
<tr>
<td>\textit{newstate}? : \textit{PROCESS_STATE}</td>
</tr>
<tr>
<td>\textit{processlogelement} : \textit{ProcessLogElement}</td>
</tr>
<tr>
<td>\textit{process} : \textit{PROCESS}</td>
</tr>
<tr>
<td>\textit{logelement} : \textit{LogElement}</td>
</tr>
<tr>
<td>\textit{r!} : \textit{Response}</td>
</tr>
</tbody>
</table>

\[
\forall p:processlist . p.process\_id \neq process\_id?
\]

\[
\textit{r!} = \textit{ProcessIDNotExists}
\]

\[
(\forall \textit{owner}: \textit{USER} . (\forall \textit{filename}: \textit{FILENAME} . \\
(\textit{owner}, \textit{filename}, \textit{Change\_Process}) \in \textit{processlogpolicy}) \lor \\
(\textit{processlogelement}.action = \textit{Change\_Process} \land \\
\textit{logelement}.r = \textit{r!} \land \textit{logelement}.u = \emptyset \land \textit{logelement}.f = \emptyset \land \\
\textit{logelement}.p = \{\textit{processlogelement}\} \land \textit{log}' = \textit{log}^{-1}(\textit{logelement}))
\]

\[
\textit{ChangeProcess} \equiv \textit{ChangeProcess\_OK} \lor \textit{ChangeProcess\_ProcessIDNotExists}
\]
APPENDIX H

Combined Authorization and Authentication

System Model Specification

[USER]
[FILENAME]
[FILE_PROPERTIES]
[PROCESS_ID]
[PROCESS_STATE]
[AUTHENTICATION_CONTROL]
Response ::= OK|UserExists|UserNotExists|FileExists|FileNotExists|
                     ProcessIDNotExists|IncorrectRights|UserNamesInconsistent|
                     InvalidUserAuthenticationControlPair|UserRunningProcess|
                     FileBeingExecuted|UserStillAuthenticated|UserStillAuthorized|
                     FileExecutionStillAuthorized
UserAuthorization ::= Add_User|Delete_User
AuthenticationAuthorization ::= Add_Authentication|Delete_Authentication
FileAuthorization ::= Add_File|Delete_File|Read_File|Change_File
ProcessAuthorization ::= Start_Process|Stop_Process|Change_Process
ACMAuthorization ::= User|File|Process|Authentication|ACM

PROCESS
filename : FILENAME
owner : USER
process_id : PROCESS_ID
state : PROCESS_STATE
\_SYSTEM

filelist : FILENAME → FILE\_PROPERTIES
userlist : \P USER
processlist : \P PROCESS
authenticated : USER → \P AUTHENTICATION\_CONTROL
userACM : USER × USER → \P UserAuthorization
fileACM : USER × FILENAME → \P FileAuthorization
processACM : (USER × (USER × FILENAME)) → \P ProcessAuthorization
authenticationACM : USER × USER → \P AuthenticationAuthorization
acmACM : USER → \P ACMAuthorization

∃p, \ p1 : processlist ∙ p1.process_id = p2.process_id ↔ p1 = p2

∀p : processlist × p.filename ∈ dom filelist ∧ p.owner ∈ userlist
dom authenticated ⊆ userlist
dom (dom userACM) ⊆ userlist
dom (dom fileACM) ⊆ userlist
dom (dom processACM) ⊆ userlist
dom (ran (dom processACM)) ⊆ userlist
ran (ran (dom processACM)) ⊆ dom filelist
dom (dom authenticationACM) ⊆ userlist
dom acmACM ⊆ userlist

Authenticate\_User

\_SYSTEM
u? : USER
ac? : AUTHENTICATION\_CONTROL
r! : Response

ac? ∈ authenticated(u?)
r! = OK
\texttt{Authenticate\_User\_Fail}\hspace{1em}

\texttt{\exists SYSTEM}
\texttt{\ u?: USER}
\texttt{\ ac?: AUTHENTICATION\_CONTROL}
\texttt{\ r!: Response}

\texttt{ac?: \in \ authenticated(u?)}
\texttt{r! = InvalidUserAuthenticationControlPair}

\texttt{Init\_SYSTEM}\hspace{1em}

\texttt{\system : SYSTEM}

\texttt{\#system.filelist = 1}
\texttt{\#system.authenticated = 1}
\texttt{\#system.userlist = 1}
\texttt{\#system.processlist = 1}
\texttt{\#system.processACM = 1}

\texttt{Add\_ACMACM\_OK}\hspace{1em}

\texttt{\triangle SYSTEM}
\texttt{\ user?: USER}
\texttt{\ newUser?: USER}
\texttt{\ rights?: \texttt{P\_ACM\_Authorization}}
\texttt{\ r!: Response}

\texttt{ACM \in acmACM(user?)}
\texttt{newUser? \in userlist}
\texttt{acmACM' = acmACM \cup \{newUser?\rightarrow rights\}}
\texttt{r! = OK}
AddACMACM_UserNotExists

\[\exists \text{SYSTEM} \quad \text{user}?: \text{USER} \quad \text{newUser}?: \text{USER} \quad \text{rights}?: \text{PACMAuthorization} \quad r! : \text{Response} \]

\[ACM \in \text{acmACM}(\text{user}) \quad \text{newUser} \notin \text{userlist} \quad r! = \text{UserNotExists} \]

AddACMACM_IncorrectRights

\[\exists \text{SYSTEM} \quad \text{user}?: \text{USER} \quad \text{newUser}?: \text{USER} \quad \text{rights}?: \text{PACMAuthorization} \quad r! : \text{Response} \]

\[ACM \in \text{acmACM}(\text{user}) \quad r! = \text{IncorrectRights} \]

\[\text{AddACMACM} \equiv (\text{Authenticate}_\text{User} \land (\text{AddACMACM}_\text{OK} \lor \text{AddACMACM}_\text{UserNotExists} \lor \text{AddACMACM}_\text{IncorrectRights})) \lor \text{Authenticate}_\text{User}_\text{Fail} \]

DeleteACMACM_OK

\[\Delta \text{SYSTEM} \quad \text{user}?: \text{USER} \quad \text{deleteUser}?: \text{USER} \quad r! : \text{Response} \]

\[ACM \in \text{acmACM}(\text{user}) \quad \text{deleteUser} \in \text{userlist} \quad \text{acmACM}' = \text{acmACM} \setminus \{\text{deleteUser}\rightarrow \text{acmACM} (\text{deleteUser})\} \quad r! = \text{OK} \]
\_DeleteACMACM\_UserNotExists

\[\exists \text{SYSTEM}\]
\[
\text{user? : USER}\hspace{1cm}
\text{deleteUser? : USER}\hspace{1cm}
\text{r! : Response}\hspace{1cm}
\]
\[
\text{ACM} \in \text{acmACM}(\text{user?})\hspace{1cm}
\text{deleteUser?} \notin \text{userlist}\hspace{1cm}
\text{r!} = \text{UserNotExists}\hspace{1cm}
\]

\_DeleteACMACM\_IncorrectRights

\[\exists \text{SYSTEM}\]
\[
\text{user? : USER}\hspace{1cm}
\text{deleteUser? : USER}\hspace{1cm}
\text{r! : Response}\hspace{1cm}
\]
\[
\text{ACM} \notin \text{acmACM}(\text{user?})\hspace{1cm}
\text{r!} = \text{IncorrectRights}\hspace{1cm}
\]

\_DeleteACMACM\_OK \lor \_DeleteACMACM\_UserNotExists \lor \_DeleteACMACM\_IncorrectRights \lor \text{Authenticate\_User\_Fail}

\_DeleteACMACM\_OK \equiv (\text{Authenticate\_User} \land (\_DeleteACMACM\_OK \lor
\_DeleteACMACM\_UserNotExists \lor
\_DeleteACMACM\_IncorrectRights)) \lor \text{Authenticate\_User\_Fail}
AddUserACM_OK

\(\Delta\text{SYSTEM} \quad \text{user}?: \text{USER} \quad \text{newUser}?: \text{USER} \quad \text{addUser}?: \text{USER} \quad \text{rights}?: \mathcal{P}\text{UserAuthorization} \quad r! : \text{Response}\)

\(\text{User} \in \text{acmACM(user?)} \quad \text{newUser} \in \text{userlist} \quad \text{userACM} = \text{userACM} \cup \{(\text{newUser},\text{addUser})\rightarrow \text{rights}\} \quad r! = \text{OK}\)

AddUserACM_IncorrectRights

\(\exists\text{SYSTEM} \quad \text{user}?: \text{USER} \quad \text{newUser}?: \text{USER} \quad \text{addUser}?: \text{USER} \quad \text{rights}?: \mathcal{P}\text{UserAuthorization} \quad r! : \text{Response}\)

\(\text{User} \in \text{acmACM(user?)} \quad r! = \text{IncorrectRights}\)

AddUserACM_NewUserNotExists

\(\exists\text{SYSTEM} \quad \text{user}?: \text{USER} \quad \text{newUser}?: \text{USER} \quad \text{addUser}?: \text{USER} \quad \text{rights}?: \mathcal{P}\text{UserAuthorization} \quad r! : \text{Response}\)

\(\text{User} \in \text{acmACM(user?)} \quad \text{newUser} \notin \text{userlist} \quad r! = \text{UserNotExists}\)
AddUserACM ≡ (Authenticate_User \ (AddUserACM_OK \ (AddUserACM_IncorrectRights \ (AddUserACM_NewUserNotExists)) \ (Authenticate_User_Fail))

_DeleteUserACM_OK

\(Δ\)SYSTEM
user? : USER
newUser? : USER
addUser? : USER
r! : Response

User ∈ acmACM(user?)
newUser? ∈ userlist
userACM' = userACM \ { (newUser?, addUser?) \→
userACM(newUser?, addUser?) }

r! = OK

_DeleteUserACM_IncorrectRights

\(∃\)SYSTEM
user? : USER
newUser? : USER
addUser? : USER
r! : Response

User ∈ acmACM(user?)

r! = IncorrectRights
\_DeleteUserACM\_NewUserNotExists

\[ \Xi \text{SYSTEM} \]
user? : USER
newUser? : USER
addUser? : USER
r! : Response

User \in \text{acmACM}(user?)
newUser? \notin \text{userlist}
r! = \text{UserNotExists}

\text{DeleteUserACM} \equiv (\text{Authenticate}_\text{User} \land (\text{DeleteUserACM\_OK} \lor 
\text{DeleteUserACM\_IncorrectRights} \lor 
\text{DeleteUserACM\_NewUserNotExists}) \lor 
\text{Authenticate}_\text{User\_Fail})

\text{AddAuthenticationACM\_OK}

\[ \Delta \text{SYSTEM} \]
user? : USER
newUser? : USER
addUser? : USER
rights? : \mathbb{P}\text{AuthenticationAuthorization}

\text{Authentication} \in \text{acmACM}(user?)
newUser? \in \text{userlist}
\text{authenticationACM} = \text{authenticationACM} \cup \{(\text{newUser?},\text{addUser?})\rightarrow\text{rights?}\}
r! = \text{OK}
\textit{AddAuthenticationACM\_IncorrectRights}

\begin{verbatim}
\exists SYSTEM
user?: USER
newUser?: USER
addUser?: USER
rights?: [AuthenticationAuthorization]
\textcolor{red}{r! : Response}

\textcolor{blue}{Authentication \in/acmACM(user?)}
\textcolor{red}{r! = IncorrectRights}
\end{verbatim}

\textit{AddAuthenticationACM\_NewUserNotExists}

\begin{verbatim}
\exists SYSTEM
user?: USER
newUser?: USER
addUser?: USER
rights?: [AuthenticationAuthorization]
\textcolor{red}{r! : Response}

\textcolor{blue}{Authentication \in/acmACM(user?)}
\textcolor{red}{newUser? \notin userlist}
\textcolor{red}{r! = UserNotExists}
\end{verbatim}

\textit{AddAuthenticationACM \equiv (Authenticate\_User \land (AddAuthenticationACM\_OK \lor AddAuthenticationACM\_IncorrectRights \lor AddAuthenticationACM\_NewUserNotExists)) \lor Authenticate\_User\_Fail}
\_DeleteAuthenticationACM\_OK

\[\text{\Delta SYSTEM}\]
\[\text{user? : USER}\]
\[\text{newUser? : USER}\]
\[\text{addUser? : USER}\]
\[\text{r! : Response}\]
\[\text{Authentication} \in \text{acmACM (user?)}\]
\[\text{newUser?} \in \text{userlist}\]
\[\text{authenticationACM'} = \text{authenticationACM} \setminus \{(\text{newUser?, addUser?})\} \rightarrow \]
\[\text{authenticationACM(newUser?, addUser?)}\]
\[\text{r!} = \text{OK}\]

\_DeleteAuthenticationACM\_IncorrectRights

\[\text{\exists SYSTEM}\]
\[\text{user? : USER}\]
\[\text{newUser? : USER}\]
\[\text{addUser? : USER}\]
\[\text{r! : Response}\]
\[\text{Authentication} \in \text{acmACM (user?)}\]
\[\text{r!} = \text{IncorrectRights}\]

\_DeleteAuthenticationACM\_NewUserNotExists

\[\text{\exists SYSTEM}\]
\[\text{user? : USER}\]
\[\text{newUser? : USER}\]
\[\text{addUser? : USER}\]
\[\text{r! : Response}\]
\[\text{Authentication} \in \text{acmACM (user?)}\]
\[\text{newUser?} \in \text{userlist}\]
\[\text{r!} = \text{UserNotExists}\]
DeleteAuthenticationACM \equiv (Authenticate\_User \land (DeleteAuthenticationACM\_OK \\
\lor DeleteAuthenticationACM\_IncorrectRights \lor \\
DeleteAuthenticationACM\_NewUserNotExists)) \lor \\
Authenticate\_User\_Fail

\textbf{AddFileACM\_OK}

\begin{align*}
ΔSYSTEM \\
user? : USER \\
newUser? : USER \\
file? : FILENAME \\
rights? : \mathcal{P}\mathit{FileAuthorization} \\
r! : Response
\end{align*}

\begin{align*}
File & \in acmACM(user?) \\
\text{newUser?} & \in \text{userlist} \\
fileACM' & = fileACM \cup \{(\text{newUser?}, file?)\rightarrow \text{rights?}\} \\
r! & = OK
\end{align*}

\textbf{AddFileACM\_IncorrectRights}

\begin{align*}
\exists SYSTEM \\
user? : USER \\
newUser? : USER \\
file? : FILENAME \\
righ\text{ts?} : \mathcal{P}\mathit{FileAuthorization} \\
r! : Response
\end{align*}

\begin{align*}
File & \in acmACM(user?) \\
r! & = IncorrectRights
\end{align*}
\[ \text{AddFileACM} \quad \text{UserNotExists} \]
\[
\begin{align*}
\text{SYSTEM} \\
\text{user? : USER} \\
\text{newUser? : USER} \\
\text{file? : FILENAME} \\
\text{rights? : PFileAuthorization} \\
\text{r! : Response} \\
\text{File} \in \text{acmACM}(\text{user?}) \\
\text{newUser?} \notin \text{userlist} \\
\text{r!} = \text{UserNotExists}
\end{align*}
\]

\[ \text{AddFileACM} \equiv (\text{Authenticate User} \land (\text{AddFileACM_OK} \lor \text{AddFileACM_IncorrectRights} \lor \text{AddFileACM_UserNotExists})) \lor \text{Authenticate User Fail} \]

\[ \text{DeleteFileACM_OK} \]
\[
\begin{align*}
\text{SYSTEM} \\
\text{user? : USER} \\
\text{newUser? : USER} \\
\text{file? : FILENAME} \\
\text{r! : Response} \\
\text{File} \in \text{acmACM}(\text{user?}) \\
\text{newUser?} \in \text{userlist} \\
\text{fileACM} = \text{fileACM} \setminus \{(\text{newUser?}\!,\text{file?})\rightarrow \text{fileACM(\text{newUser?}\!,\text{file?]})}\} \\
\end{align*}
\]
\texttt{DeleteFileACM\_IncorrectRights}
\begin{verbatim}
\texttt{\exists SYSTEM}
user? : USER
newUser? : USER
file? : FILENAME
r! : Response

File \in \texttt{acmACM(user?)}
r! = IncorrectRights
\end{verbatim}

\texttt{DeleteFileACM\_UserNotExists}
\begin{verbatim}
\texttt{\exists SYSTEM}
user? : USER
newUser? : USER
file? : FILENAME
r! : Response

File \in \texttt{acmACM(user?)}
newUser? \notin \texttt{userlist}
r! = UserNotExists
\end{verbatim}

\texttt{DeleteFileACM} \equiv (\texttt{Authenticate\_User} \land (\texttt{DeleteFileACM\_OK} \lor \\
\texttt{DeleteFileACM\_IncorrectRights} \lor \\
\texttt{DeleteFileACM\_UserNotExists}) \lor \texttt{Authenticate\_User\_Fail}
**AddProcessACM_OK**

\( \Delta \text{SYSTEM} \)

\( \text{user}? : \text{USER} \)

\( \text{newUser}? : \text{USER} \)

\( \text{runAs}? : \text{USER} \)

\( \text{file}? : \text{FILENAME} \)

\( \text{rights}? : \mathcal{P}\text{ProcessAuthorization} \)

\( r! : \text{Response} \)

\( \text{Process } \in \text{acmACM}(\text{user}?) \)

\( \text{newUser}? \in \text{userlist} \)

\( \text{processACM} \equiv \text{processACM} \cup \{(\text{newUser}?,(\text{runAs}?,\text{file}?)\rightarrow \text{rights}?)\} \)

\( r! = \text{OK} \)

---

**AddProcessACM_IncorrectRights**

\( \equiv \text{SYSTEM} \)

\( \text{user}? : \text{USER} \)

\( \text{newUser}? : \text{USER} \)

\( \text{runAs}? : \text{USER} \)

\( \text{file}? : \text{FILENAME} \)

\( \text{rights}? : \mathcal{P}\text{ProcessAuthorization} \)

\( r! : \text{Response} \)

\( \text{Process } \in \text{acmACM}(\text{user}?) \)

\( r! = \text{IncorrectRights} \)
_AddProcessACM_UserNotExists_

\[ \exists \text{SYSTEM} \]
\[ \text{user}?: \text{USER} \]
\[ \text{newUser}?: \text{USER} \]
\[ \text{runAs}?: \text{USER} \]
\[ \text{file}?: \text{FILENAME} \]
\[ \text{rights}?: \text{ProcessAuthorization} \]
\[ r! : \text{Response} \]

\[ \text{Process} \in \text{acmACM(user?)} \]
\[ \text{newUser} \not\in \text{userlist} \]
\[ r! = \text{UserNotExists} \]

_AddProcessACM \equiv (\text{Authenticate User} \land (\text{AddProcessACM_OK} \lor \text{AddProcessACM_IncorrectRights} \lor \text{AddProcessACM_UserNotExists}) \lor \text{Authenticate User Fail} \]

_DeleteProcessACM_OK_

\[ \Delta \text{SYSTEM} \]
\[ \text{user}?: \text{USER} \]
\[ \text{newUser}?: \text{USER} \]
\[ \text{runAs}?: \text{USER} \]
\[ \text{file}?: \text{FILENAME} \]
\[ r! : \text{Response} \]

\[ \text{Process} \in \text{acmACM(user?)} \]
\[ \text{newUser} \in \text{userlist} \]
\[ \text{processACM} = \text{processACM} \setminus \{(\text{newUser?},(\text{runAs?},\text{file?}))\rightarrow \]
\[ \text{processACM(newUser?),(runAs?,file?)})\} \]
\[ r! = \text{OK} \]
\_DeleteProcessACM\_IncorrectRights

\[\exists SYSTEM\]
user? : USER
newUser? : USER
runAs? : USER
file? : FILENAME
r! : Response

Process \(\in\ acmACM\ (user?)\)
r! = IncorrectRights

\_DeleteProcessACM\_UserNotExists

\[\exists SYSTEM\]
user? : USER
newUser? : USER
runAs? : USER
file? : FILENAME
r! : Response

Process \(\in\ acmACM\ (user?)\)
newUser? \(\notin\ userlist\)
r! = UserNotExists

\_DeleteProcessACM

\(\equiv\ (Authenticate\_User \land (DeleteProcessACM\_OK \lor\)
\quad DeleteProcessACM\_IncorrectRights \lor\)
\quad DeleteProcessACM\_UserNotExists)) \lor\)
\quad Authenticate\_User\_Fail\)
AddUser_Authorization_OK

\[ \text{SYSTEM} \]
\[ \text{user? : USER} \]
\[ \text{newUser? : USER} \]
\[ r! : \text{Response} \]
\[ \text{Add\_User} \in \text{userACM(user?,newUser?)} \]
\[ r! = \text{OK} \]

AddUser_Authorization_Fail

\[ \text{SYSTEM} \]
\[ \text{user? : USER} \]
\[ \text{newUser? : USER} \]
\[ r! : \text{Response} \]
\[ \text{Add\_User} \in \text{userACM(user?,newUser?)} \]
\[ r! = \text{IncorrectRights} \]

DeleteUser_Authorization_OK

\[ \text{SYSTEM} \]
\[ \text{user? : USER} \]
\[ \text{newUser? : USER} \]
\[ r! : \text{Response} \]
\[ \text{Delete\_User} \in \text{userACM(user?,newUser?)} \]
\[ r! = \text{OK} \]

DeleteUser_Authorization_Fail

\[ \text{SYSTEM} \]
\[ \text{user? : USER} \]
\[ \text{newUser? : USER} \]
\[ r! : \text{Response} \]
\[ \text{Delete\_User} \in \text{userACM(user?,newUser?)} \]
\[ r! = \text{IncorrectRights} \]
AddAuthentication_Authorization_OK

\[\exists \text{SYSTEM} \\ \ \ \ user? : \text{USER} \ \ \ \ new\text{(newUser)? : USER} \ \ \ \ r! : \text{Response}\]

Add\_Authentication \in \text{authenticationACM}\(user?,new\text{User}\)\)

\[r! = \text{OK}\]

AddAuthentication_Authorization_Fail

\[\exists \text{SYSTEM} \\ \ \ \ user? : \text{USER} \ \ \ \ new\text{(newUser)? : USER} \ \ \ \ r! : \text{Response}\]

Add\_Authentication \in \text{authenticationACM}\(user?,new\text{User}\)\)

\[r! = \text{IncorrectRights}\]

DeleteAuthentication_Authorization_OK

\[\exists \text{SYSTEM} \\ \ \ \ user? : \text{USER} \ \ \ \ new\text{(newUser)? : USER} \ \ \ \ r! : \text{Response}\]

Delete\_Authentication \in \text{authenticationACM}\(user?,new\text{User}\)\)

\[r! = \text{OK}\]

DeleteAuthentication_Authorization_Fail

\[\exists \text{SYSTEM} \\ \ \ \ user? : \text{USER} \ \ \ \ new\text{(newUser)? : USER} \ \ \ \ r! : \text{Response}\]

Delete\_Authentication \in \text{authenticationACM}\(user?,new\text{User}\)\)

\[r! = \text{IncorrectRights}\]
AddFile_Authorization_OK

ESYSTEM
user?: USER
file?: FILENAME
r!: Response
Add_File ∈ fileACM(user?, file?)
r! = OK

AddFile_Authorization_Fail

ESYSTEM
user?: USER
file?: FILENAME
r!: Response
Add_File ∈ fileACM(user?, file?)
r! = IncorrectRights

DeleteFile_Authorization_OK

ESYSTEM
user?: USER
file?: FILENAME
r!: Response
Delete_File ∈ fileACM(user?, file?)
r! = OK

DeleteFile_Authorization_Fail

ESYSTEM
user?: USER
file?: FILENAME
r!: Response
Delete_File ∈ fileACM(user?, file?)
r! = IncorrectRights
_ReadFile_Authorization_OK_

\[ \exists \text{SYSTEM} \]
user?: USER
file?: FILENAME
r! : Response

Read_File ∈ fileACM(user?, file?)
r! = OK

_ReadFile_Authorization_Fail_

\[ \exists \text{SYSTEM} \]
user?: USER
file?: FILENAME
r! : Response

Read_File ∈ fileACM(user?, file?)
r! = IncorrectRights

_ChangeFile_Authorization_OK_

\[ \exists \text{SYSTEM} \]
user?: USER
file?: FILENAME
r! : Response

Change_File ∈ fileACM(user?, file?)
r! = OK

_ChangeFile_Authorization_Fail_

\[ \exists \text{SYSTEM} \]
user?: USER
file?: FILENAME
r! : Response

Change_File ∈ fileACM(user?, file?)
r! = IncorrectRights
StartProcess Authorization_OK

\[
\begin{align*}
\exists \text{SYSTEM} \\
\text{user?} & : \text{USER} \\
\text{runAs?} & : \text{USER} \\
\text{file?} & : \text{FILENAME} \\
\text{r!} & : \text{Response} \\
\text{Start\_Process} & \in \text{processACM}(\text{user?}, (\text{runAs?}, \text{file?})) \\
\text{r!} & = \text{OK}
\end{align*}
\]

StartProcess Authorization_Fail

\[
\begin{align*}
\exists \text{SYSTEM} \\
\text{user?} & : \text{USER} \\
\text{runAs?} & : \text{USER} \\
\text{file?} & : \text{FILENAME} \\
\text{r!} & : \text{Response} \\
\text{Start\_Process} & \in \text{processACM}(\text{user?}, (\text{runAs?}, \text{file?})) \\
\text{r!} & = \text{IncorrectRights}
\end{align*}
\]

StopProcess Authorization_OK

\[
\begin{align*}
\exists \text{SYSTEM} \\
\text{user?} & : \text{USER} \\
\text{runAs?} & : \text{USER} \\
\text{file?} & : \text{FILENAME} \\
\text{r!} & : \text{Response} \\
\text{Stop\_Process} & \in \text{processACM}(\text{user?}, (\text{runAs?}, \text{file?})) \\
\text{r!} & = \text{OK}
\end{align*}
\]
StopProcess_Authorization_Fail

\[ \exists \text{SYSTEM} \]
user? : USER
runAs? : USER
file? : FILENAME
\[ r! : \text{Response} \]

Stop_Process \in \text{processACM}(user?, (runAs?, file?))
\[ r! = \text{IncorrectRights} \]

ChangeProcess_Authorization_OK

\[ \exists \text{SYSTEM} \]
user? : USER
runAs? : USER
file? : FILENAME
\[ r! : \text{Response} \]

Change_Process \in \text{processACM}(user?, (runAs?, file?))
\[ r! = \text{OK} \]

ChangeProcess_Authorization_Fail

\[ \exists \text{SYSTEM} \]
user? : USER
runAs? : USER
file? : FILENAME
\[ r! : \text{Response} \]

Change_Process \in \text{processACM}(user?, (runAs?, file?))
\[ r! = \text{IncorrectRights} \]
AddAuthentication_OK

ΔSYSTEM
user? : USER
ac? : AUTHENTICATION_CONTROL
r! : Response

user? ∈ userlist
authenticated'(user?) = authenticated(user?) \cup \{ac?\}
r! = OK

AddAuthentication_NotExists

∃SYSTEM
user? : USER
ac? : AUTHENTICATION_CONTROL
r! : Response

user? ∉ userlist
r! = UserNotExists

AddAuthentication = \( (Authenticate_User \land ((AddAuthentication_Authorization_OK \land (AddAuthentication_OK \lor AddAuthentication_NotExists)) \lor AddAuthentication_Authorization_Fail)) \lor Authenticate_User_Fail \)

DeleteAuthentication_OK

ΔSYSTEM
user? : USER
ac? : AUTHENTICATION_CONTROL
r! : Response

ac? ∉ authenticated(user?)
authenticated'(user?) = authenticated(user?) \setminus \{ac?\}
DeleteAuthentication_NotExists

\[ \Delta \text{SYSTEM} \]
\[ \text{user?} : \text{USER} \]
\[ \text{ac?} : \text{AUTHENTICATION\_CONTROL} \]
\[ r! : \text{Response} \]
\[ \text{ac?} \notin \text{authenticated(user?)} \]
\[ r! = \text{InvalidUserAuthenticationControlPair} \]

DeleteAuthentication \equiv (\text{Authenticate\_User} \land
\left( (\text{DeleteAuthentication\_Authorization\_OK} \land
\left( (\text{DeleteAuthentication\_OK} \lor \text{DeleteAuthentication\_NotExists}) \lor \text{DeleteAuthentication\_Authorization\_Fail} \right) \lor \text{Authenticate\_User\_Fail} \right) \right) \]

AddUser\_OK

\[ \Delta \text{SYSTEM} \]
\[ \text{newUser?} : \text{USER} \]
\[ r! : \text{Response} \]
\[ \text{newUser?} \notin \text{userlist} \]
\[ \text{userlist'} = \text{userlist} \cup \{\text{newUser?}\} \]
\[ r! = \text{OK} \]

AddUser\_Exists

\[ \Delta \text{SYSTEM} \]
\[ \text{newUser?} : \text{USER} \]
\[ r! : \text{Response} \]
\[ \text{newUser?} \in \text{userlist} \]
\[ r! = \text{UserExists} \]
$$AddUser \equiv (Authenticate_{User} \land ((AddUser\_Authorization\_OK \land (AddUser\_OK \lor AddUser\_Exists)) \lor AddUser\_Authorization\_Fail)) \lor Authenticate\_User\_Fail$$

_DeleteUser\_OK_

\[\text{SYS} \text{TEM} \]
\[\begin{align*}
&\text{deleteUser? : USER} \\
&r! : \text{Response} \\
\end{align*}\]

\[\begin{align*}
&\text{deleteUser?} \in \text{userlist} \\
&\forall p : \text{processlist} \bullet p.\text{owner} \neq \text{deleteUser?} \\
&\text{deleteUser?} \in \text{dom authenticated} \\
&\text{deleteUser?} \in \text{dom (dom userACM)} \\
&\text{deleteUser?} \in \text{dom (dom fileACM)} \\
&\text{deleteUser?} \in \text{dom (dom processACM)} \\
&\text{deleteUser?} \in \text{dom (ran (dom processACM))} \\
&\text{deleteUser?} \in \text{dom acmACM} \\
&\text{userlist}' = \text{userlist} \setminus \{\text{deleteUser?}\} \\
&r! = \text{OK}
\end{align*}\]

_DeleteUser\_NotExists_

\[\text{SYS} \text{TEM} \]
\[\begin{align*}
&\text{deleteUser? : USER} \\
&r! : \text{Response} \\
\end{align*}\]

\[\begin{align*}
&\text{deleteUser?} \in \text{userlist} \\
&r! = \text{UserNotExists}
\end{align*}\]
\_DeleteUser\_RunningProcess

\(\exists \text{SYSTEM} \)
\(\text{deleteUser}\? : \text{USER} \)
\(r! : \text{Response} \)
\(\text{deleteUser}\? \in \text{userlist} \)
\(\exists p : \text{processlist} \bullet p.\text{owner} = \text{deleteUser}\? \)
\(r! = \text{UserRunningProcess} \)

\_DeleteUser\_StillAuthenticated

\(\exists \text{SYSTEM} \)
\(\text{deleteUser}\? : \text{USER} \)
\(r! : \text{Response} \)
\(\text{deleteUser}\? \in \text{userlist} \)
\(\forall p : \text{processlist} \bullet p.\text{owner} \neq \text{deleteUser}\? \)
\(\text{deleteUser}\? \in \text{dom authenticated} \)
\(r! = \text{UserStillAuthenticated} \)

\_DeleteUser\_UserStillAuthorized

\(\exists \text{SYSTEM} \)
\(\text{deleteUser}\? : \text{USER} \)
\(r! : \text{Response} \)
\(\text{deleteUser}\? \in \text{userlist} \)
\(\forall p : \text{processlist} \bullet p.\text{owner} \neq \text{deleteUser}\? \)
\(\text{deleteUser}\? \in \text{dom (dom userACM)} \lor \)
\(\text{deleteUser}\? \in \text{dom (dom fileACM)} \lor \)
\(\text{deleteUser}\? \in \text{dom (dom processACM)} \lor \)
\(\text{deleteUser}\? \in \text{dom (ran (dom processACM))} \lor \)
\(\text{deleteUser}\? \in \text{dom acmACM} \)
\(r! = \text{UserStillAuthorized} \)
\[ \text{DeleteUser} \equiv (\text{Authenticate\_User} \land ((\text{DeleteUser\_Authorization\_OK} \land \\
\quad \text{DeleteUser\_OK} \lor \text{DeleteUser\_NotExists} \lor \\
\quad \text{DeleteUser\_RunningProcess} \lor \text{DeleteUser\_StillAuthenticated} \lor \\
\quad \text{DeleteUser\_UserStillAuthorized})) \lor \text{DeleteUser\_Authorization\_Fail})) \\
\lor \text{Authenticate\_User\_Fail} \]

\[ \text{AddFile\_OK} \]
\[ \Delta \text{SYSTEM} \]
\[ \text{filename}\?: \text{FILENAME} \]
\[ \text{fileproperties}\?: \text{FILE\_PROPERTIES} \]
\[ r! : \text{Response} \]
\[ \text{filename}\? \in \text{dom\_filelist} \]
\[ \text{filelist}' = \text{filelist} \cup \{\text{filename}\? \mapsto \text{fileproperties}\?\} \]
\[ r! = \text{OK} \]

\[ \text{AddFile\_Exists} \]
\[ \exists \text{SYSTEM} \]
\[ \text{filename}\?: \text{FILENAME} \]
\[ \text{fileproperties}\?: \text{FILE\_PROPERTIES} \]
\[ r! : \text{Response} \]
\[ \text{filename}\? \in \text{dom\_filelist} \]
\[ r! = \text{FileExists} \]

\[ \text{AddFile} \equiv (\text{Authenticate\_User} \land ((\text{AddFile\_Authorization\_OK} \land \\
\quad \text{AddFile\_OK} \lor \text{AddFile\_Exists})) \lor \text{AddFile\_Authorization\_Fail})) \lor \text{Authenticate\_User\_Fail} \]
_ReadFile_OK_

\[\exists \text{SYSTEM} \]
\[\text{filename?} : \text{FILENAME}\]
\[\text{fileproperties!} : \text{FILE\_PROPERTIES}\]
\[r! : \text{Response}\]

\[\text{filename?} \in \text{dom filelist}\]
\[\text{fileproperties!} = \text{filelist}(\text{filename?})\]
\[r! = \text{OK}\]

_ReadFile_NotExists_

\[\exists \text{SYSTEM} \]
\[\text{filename?} : \text{FILENAME}\]
\[\text{fileproperties!} : \text{FILE\_PROPERTIES}\]
\[r! : \text{Response}\]

\[\text{filename?} \in \text{dom filelist}\]
\[r! = \text{FileNotExists}\]

\[
\text{ReadFile} \equiv (\text{Authenticate\_User} \land ((\text{ReadFile\_Authorization\_OK} \land
\text{ReadFile\_OK} \lor \text{ReadFile\_NotExists})) \lor
\text{ReadFile\_Authorization\_Fail}) \lor \text{Authenticate\_User\_Fail}\]

_DeleteFile_OK_

\[\Delta \text{SYSTEM} \]
\[\text{filename?} : \text{FILENAME}\]
\[r! : \text{Response}\]

\[\text{filename?} \in \text{dom filelist}\]
\[\forall p : \text{processlist} \bullet p.\text{filename} \neq \text{filename}?\]
\[\text{filename?} \notin \text{ran (ran (dom processACM))}\]
\[\text{filelist'} = \text{filelist} \setminus \{\text{filename}? \rightarrow \text{filelist}(\text{filename}?)\}\]
\[r! = \text{OK}\]
DeleteFile_NotExists

\(\exists \text{SYSTEM} \quad \text{filename}?: \text{FILENAME} \quad \text{r!} : \text{Response} \)

\(\text{filename}? \in \text{dom filelist} \quad \text{r!} = \text{FileNotExists} \)

DeleteFile_FileExecutionStillAuthorized

\(\exists \text{SYSTEM} \quad \text{filename}?: \text{FILENAME} \quad \text{r!} : \text{Response} \)

\(\text{filename}? \in \text{dom filelist} \quad \forall p : \text{processlist} \quad p.\text{filename} \neq \text{filename}? \quad \text{filename}? \in \text{ran} (\text{ran} (\text{dom processACM})) \quad \text{r!} = \text{FileExecutionStillAuthorized} \)

DeleteFile_FileBeingExecuted

\(\exists \text{SYSTEM} \quad \text{filename}?: \text{FILENAME} \quad \text{r!} : \text{Response} \)

\(\text{filename}? \in \text{dom filelist} \quad \exists p : \text{processlist} \quad p.\text{filename} = \text{filename}? \quad \text{r!} = \text{FileBeingExecuted} \)

DeleteFile \equiv (\text{Authenticate User} \land ((\text{DeleteFile Authorization OK} \land
\quad (\text{DeleteFile OK} \lor \text{DeleteFile NotExists} \lor \text{DeleteFile FileBeingExecuted} \lor \text{DeleteFile FileExecutionStillAuthorized}) \lor \text{DeleteFile Authorization Fail}))) \lor \text{Authenticate User Fail}
\textbf{ChangeFile\_OK}
\begin{verbatim}
\textbf{SYSTEM}
\textbf{filename}\,: FILENAME
\textbf{properties}\,: FILE\textunderscore PROPERTIES
\textbf{r!}\,: \textbf{Response}

\textbf{filename}\,\in\,\textbf{dom\,filelist}
\textbf{filelist}'\,=\,\textbf{filelist} \cup \{\textbf{filename}\,\mapsto\,\textbf{properties}\}\}
\textbf{r!}\,=\,\textbf{OK}
\end{verbatim}

\textbf{ChangeFile\_NotExists}
\begin{verbatim}
\textbf{SYSTEM}
\textbf{filename}\,: FILENAME
\textbf{properties}\,: FILE\textunderscore PROPERTIES
\textbf{r!}\,: \textbf{Response}

\textbf{filename}\,\not\in\,\textbf{dom\,filelist}
\textbf{r!}\,=\,\textbf{FileNotExists}
\end{verbatim}

\textbf{ChangeFile} \equiv (\textbf{Authenticate\_User} \land ((\textbf{ChangeFile\_Authorization\_OK} \land
\textbf{ChangeFile\_OK} \lor \textbf{ChangeFile\_NotExists}) \lor
\textbf{ChangeFile\_Authorization\_Fail}) \lor \textbf{Authenticate\_User\_Fail})

\textbf{StartProcess\_OK}
\begin{verbatim}
\textbf{SYSTEM}
\textbf{process}\,: PROCESS
\textbf{runAs}\,: USER
\textbf{r!}\,: \textbf{Response}

\textbf{process}\,._{owner}\,=\,\textbf{runAs}\?
\textbf{process}\,._{filename}\,\in\,\textbf{dom\,filelist}
\forall\textbf{p\,process\,list}\,\bullet\textbf{p\,process\,id}\,\ne\,\textbf{process}\,._{process\,id}
\textbf{processlist}'\,=\,\textbf{processlist} \cup \{\textbf{process}\,\}
\textbf{r!}\,=\,\textbf{OK}
\end{verbatim}
\textit{StartProcess\_FileNotFoundException}

\begin{align*}
\exists \text{SYSTEM} \\
\text{process? : PROCESS} \\
\text{runAs? : USER} \\
\text{r! : Response} \\
\text{process?\_filename} \in \text{dom\_filelist} \\
r! = \text{FileNotFoundException}
\end{align*}

\textit{StartProcess\_UserNamesInconsistent}

\begin{align*}
\exists \text{SYSTEM} \\
\text{process? : PROCESS} \\
\text{runAs? : USER} \\
\text{r! : Response} \\
\text{process?\_owner} \neq \text{runAs?} \\
r! = \text{UserNamesInconsistent}
\end{align*}

\textit{StartProcess} \equiv (\text{Authenticate\_User} \land ((\text{StartProcess\_Authorization\_OK} \land \\
(\text{StartProcess\_OK} \lor \text{StartProcess\_FileNotFoundException} \\
\lor \text{StartProcess\_UserNamesInconsistent})) \lor \\
\text{StartProcess\_Authorization\_Fail}) \lor \text{Authenticate\_User\_Fail})

\textit{StopProcess\_OK}

\begin{align*}
\Delta \text{SYSTEM} \\
\text{process\_id? : PROCESS\_ID} \\
\text{runAs? : USER} \\
\text{r! : Response} \\
\exists p: \text{processlist} \bullet p\_process\_id = \text{process\_id?} \land p\_owner = \text{runAs?} \\
\text{processlist}! = \text{processlist}\setminus\{p: \text{PROCESS}\{p\_process\_id = \text{process\_id?}\}\} \\
r! = \text{OK}
\end{align*}
\[ \text{StopProcess\_ProcessIDNotExists} \]\\
\[ \exists \text{SYSTEM} \]
\[ \text{process\_id? : PROCESS\_ID} \]
\[ \text{runAs? : USER} \]
\[ \text{r! : Response} \]
\[ \forall p: \text{processlist} \land p.\text{process\_id} \neq \text{process\_id?} \]
\[ \text{r!} = \text{ProcessIDNotExists} \]

\[ \text{StopProcess\_UserNamesInconsistent} \]\\
\[ \exists \text{SYSTEM} \]
\[ \text{process\_id? : PROCESS\_ID} \]
\[ \text{runAs? : USER} \]
\[ \text{r! : Response} \]
\[ \exists p: \text{processlist} \land p.\text{process\_id} = \text{process\_id?} \]
\[ \forall p: \text{processlist} \land p.\text{owner} \neq \text{runAs?} \]
\[ \text{r!} = \text{UserNameInconsistent} \]

\[ \text{StopProcess} \equiv (\text{Authenticate\_User} \land ((\text{StopProcess\_Authorization\_OK} \land
(\text{StopProcess\_OK} \lor \text{StopProcess\_ProcessIDNotExists} \lor
\text{StopProcess\_UserNamesInconsistent})) \lor
\text{StopProcess\_Authorization\_Fail}) \lor \text{Authenticate\_User\_Fail} \]

\[ \text{ChangeProcess\_OK} \]\\
\[ \Delta \text{SYSTEM} \]
\[ \text{process\_id? : PROCESS\_ID} \]
\[ \text{newstate? : PROCESS\_STATE} \]
\[ \text{r! : Response} \]
\[ (\text{let } p \equiv \{ p1: \text{processlist} \mid p1.\text{process\_id} = \text{process\_id}\}\bullet \]
\[ (\forall p2: p \land p2.\text{state} = \text{newstate}\}\land \# p = 1 \land \]
\[ \text{processlist}' = \text{processlist} \setminus \{ p2: \text{processlist} \mid p2.\text{process\_id} = \text{process\_id}\} \land \]
\[ \text{processlist}' = \text{processlist}' \cup p \land \]
\[ \text{r!} = \text{OK} \]
ChangeProcess \_ProcessIDNotExists

\[\exists \text{SYSTEM}
\text{process\_id}? : \text{PROCESS\_ID}
\text{newstate}? : \text{PROCESS\_STATE}
\text{r!} : \text{Response}
\]

\[\forall p: \text{processlist} \bullet p\text{.process\_id} \neq \text{process\_id}?
\text{r!} = \text{ProcessIDNotExists}
\]

ChangeProcess \equiv (\text{Authenticate\_User} \land ((\text{ChangeProcess\_Authorization\_OK} \land
(\text{ChangeProcess\_OK} \lor \text{ChangeProcess\_ProcessIDNotExists})) \lor
\text{ChangeProcess\_Authorization\_Fail})) \lor \text{Authenticate\_User\_Fail}
REFERENCES


