Especificación Formal de un Sistema de Control de Acceso para Laboratorios en la Universidad La Salle de Arequipa

Jhordan Huamaní Huamaní[1], Jorge Ortiz Castañeda[1], José Mamani Zuñiga[1], Miguel Flores León[1]
[1]Departamento de Ingeniería de Software, Universidad La Salle de Arequipa, Perú
{jhordanhh, jortizc, jmamamniz, mfloresl}@ulasalle.edu.pe

***Abstract*—Este artículo presenta el desarrollo del proyecto *Especificación Formal de un Sistema de Control de Acceso para Laboratorios en la Universidad La Salle de Arequipa*, realizado por estudiantes de la Universidad La Salle y expuesto en la Feria de Proyectos *La Salle In-Genio* 2025. El objetivo principal es diseñar la especificación formal del sistema (SICA-L) utilizando VDM++, abordando la vulnerabilidad de los equipos de alto valor y la actual falta de trazabilidad. La propuesta consiste en validar el modelo mediante herramientas como VDM++ Toolbox y NuSMV, destacando su contribución en términos de seguridad preventiva y la generación de un diseño libre de ambigüedades. Finalmente, se discuten los resultados preliminares y el potencial de aplicación en futuras implementaciones para instituciones de educación superior.**

***Index Terms*—Especificación formal, Modelos de control de acceso, Sistema de control de acceso, Verificación de modelos**

I. Introduction

La Universidad La Salle de Arequipa ha emprendido un proceso de modernización de su infraestructura tecnológica y académica, incorporando laboratorios con equipamiento de alto valor. En este contexto, surge la necesidad crítica de implementar un sistema que permita controlar, registrar y auditar los accesos a dichos espacios de manera formal y verificable. La gestión eficiente de estos recursos exige soluciones que garanticen no solo la seguridad física, sino también la trazabilidad inequívoca de las operaciones realizadas por el personal autorizado.

Los sistemas tradicionales de control, basados frecuentemente en registros manuales o mecanismos informales, resultan insuficientes ante los estándares actuales de seguridad. Esta carencia evidencia la vulnerabilidad ante errores humanos y dificulta la auditoría forense en caso de incidentes. Por consiguiente, resulta imperativo diseñar un mecanismo automatizado que incorpore criterios formales para eliminar ambigüedades, inconsistencias lógicas y brechas de seguridad que podrían comprometer la integridad institucional.

El presente artículo introduce el proyecto *Especificación Formal de un Sistema de Control de Acceso para Laboratorios (SICA-L)*, desarrollado por estudiantes de la Universidad La Salle y expuesto en la Feria de Proyectos *La Salle In-Genio 2025*. La propuesta central consiste en la aplicación de métodos formales, específicamente el lenguaje VDM++, para modelar las operaciones críticas del sistema tales como autenticación, autorización y registro asegurando la corrección matemática del diseño antes de su codificación.

Más allá de la especificación, el modelo es sometido a una rigurosa validación y verificación: se emplea análisis de cobertura para la lógica interna y herramientas de *model checking* como NuSMV y UPPAAL para garantizar el cumplimiento de propiedades de seguridad y restricciones de tiempo real. De este modo, el trabajo establece un *blueprint* formal y robusto, sirviendo como referencia replicable para futuras implementaciones en entornos académicos y de alta seguridad.

Presentado en la Feria de Proyectos *Inspira La Salle*, Universidad La Salle, Arequipa, 2025

II. Trabajos Anteriores

La aplicación de métodos formales para la especificación y verificación de sistemas de control de acceso es un campo de investigación bien establecido, ya que la seguridad y la fiabilidad son requisitos no negociables en estos sistemas. El presente trabajo se sitúa en la intersección de varias áreas de investigación consolidadas.

El uso del *Vienna Development Method* (VDM) y su extensión orientada a objetos, VDM++, para modelar sistemas críticos tiene una larga trayectoria. Autores como Bryans y Fitzgerald han demostrado cómo VDM++ puede ser utilizado para la ingeniería formal de políticas de control de acceso complejas, como las expresadas en XACML, permitiendo un análisis riguroso antes de la implementación [1]. La versatilidad de VDM++ también se ha demostrado en la especificación de sistemas industriales, como los sistemas de autodefensa para aeronaves de combate, donde la precisión del modelo es fundamental para garantizar la seguridad operacional [2]. Además, su aplicación se extiende a sistemas de transacciones seguras, como monederos electrónicos, donde la especificación formal ha sido clave para detectar errores sutiles en el código fuente [3].

El *model checking* es la técnica de verificación por excelencia para encontrar fallos en sistemas concurrentes y de seguridad [4]. Herramientas como NuSMV y SPIN se han utilizado extensamente para analizar protocolos de seguridad y encontrar vulnerabilidades que las pruebas manuales no logran detectar, como en el famoso caso del protocolo Needham-Schroeder [5]. Se ha empleado, por ejemplo, para validar políticas de control de acceso expresadas en SecureUML, traduciéndolas a un lenguaje formal como B para realizar pruebas

sistemáticas de permisos de roles [6]. La formalización de propiedades de seguridad como autorización, autenticación, integridad y confidencialidad utilizando VDM-SL, el precursor de VDM++, ha demostrado la capacidad de estos métodos para especificar sistemas sin ambigüedades [7].

En el ámbito de los sistemas de tiempo real, donde las restricciones temporales son tan importantes como la corrección lógica, UPPAAL es la herramienta de referencia [8]. Se ha aplicado con éxito para verificar sistemas críticos como los controladores de semáforos inteligentes, donde un fallo en la temporización puede tener consecuencias catastróficas [9]. Su capacidad para modelar y verificar protocolos con restricciones de tiempo también lo hace adecuado para analizar aspectos temporales en las políticas de control de acceso, como las definidas en el modelo *Temporal Role-Based Access Control* (TRBAC) [10].

Este trabajo se distingue por su enfoque metodológico integral: no solo se especifica el sistema de control de acceso en VDM++, sino que se valida su lógica interna con análisis de cobertura y, crucialmente, se verifica formalmente con un doble enfoque de *model checking*. Se utiliza NuSMV para probar la robustez de las políticas de seguridad desde una perspectiva lógica y atemporal, y UPPAAL para garantizar que el comportamiento del sistema cumple con las restricciones de tiempo del mundo real. Esta combinación de técnicas proporciona un nivel de confianza en la correctitud del sistema que es difícil de alcanzar con un único método.

III. Objetivos

A. Objetivo General

Elaborar la especificación formal de un sistema de control de acceso para los nuevos laboratorios de la Universidad La Salle, que garantice la seguridad y la trazabilidad desde su puesta en marcha, y que sirva como modelo base escalable para otras instituciones de educación superior.

B. Objetivos Específicos

1) Modelar las entidades clave del sistema: Usuarios, Laboratorios y Permisos de acceso.
2) Especificar las políticas y reglas de acceso mediante un método formal para lograr la precisión necesaria en el diseño.
3) Definir las operaciones críticas (como la verificación de acceso y gestión de permisos) y establecer los invariantes del sistema para asegurar su consistencia lógica.

IV. Metodología

La metodología empleada en este trabajo es de naturaleza computacional y formal, basada en el ciclo de vida de desarrollo de software riguroso para sistemas críticos. El proceso se centra en la transformación sistemática de requerimientos informales en especificaciones matemáticas precisas, permitiendo la detección temprana de ambigüedades y errores lógicos.

El desarrollo del proyecto se ha estructurado en cuatro etapas secuenciales e iterativas, tal como se ilustra en el diagrama de flujo de la Fig. 1, y se detalla a continuación:

A. Análisis y Abstracción de Requerimientos

En esta etapa inicial se realizó la recopilación de datos mediante la observación de los procesos actuales de acceso en los laboratorios de la Universidad La Salle. Se identificaron los actores (estudiantes, docentes, administrativos), los recursos (laboratorios) y las restricciones operativas.

- **Entrada:** Reglas de negocio en lenguaje natural y entrevistas con los responsables de laboratorio.
- **Salida:** Lista refinada de Requerimientos Funcionales (RF) y un Diagrama de Clases UML preliminar.

B. Especificación Formal con VDM++

Esta fase constituye el núcleo del modelado teórico. Se procedió a la formalización de la estructura estática y dinámica del sistema utilizando el lenguaje *Vienna Development Method* orientado a objetos (VDM++). Se definieron tres componentes matemáticos clave:

1) **Tipos de Datos:** Abstracción de entidades mediante `types` personalizados.
2) **Invariantes de Estado:** Ecuaciones lógicas que restringen los valores permitidos de las variables de instancia para mantener la integridad del sistema.

C. Validación Interna y Análisis de Cobertura

Para garantizar la consistencia interna del modelo, se empleó una metodología experimental de pruebas basadas en escenarios. Utilizando la herramienta *VDM++ Toolbox*.

El criterio de éxito establecido fue alcanzar un 100% de cobertura de código (*statement coverage*), asegurando que todas las líneas de la especificación, incluyendo las cláusulas de manejo de excepciones, fueran ejercitadas durante la simulación.

D. Verificación de Modelos (Model Checking)

Finalmente, se realizó la verificación externa de propiedades críticas mediante técnicas de *Model Checking*, dividiendo el análisis en dos dominios complementarios:

1) Seguridad Lógica (NuSMV): Se tradujo el modelo a una máquina de estados finitos para verificar propiedades de seguridad (*safety properties*) expresadas en Lógica de Árbol Computacional (CTL), asegurando matemáticamente que nunca ocurra un estado prohibido.

2) Tiempo Real (UPPAAL): Se modeló el sistema como una red de autómatas temporizados para validar restricciones temporales (*liveness properties*), verificando que el sistema responda a las solicitudes de acceso dentro de los plazos establecidos (latencia máxima permitida).

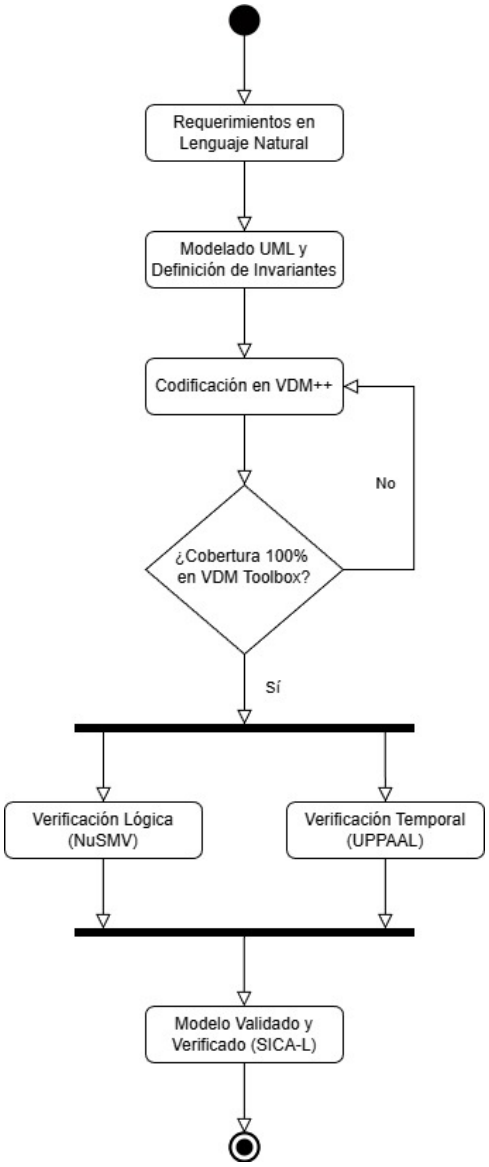

Fig. 1: Diagrama de flujo de la metodología

V. Desarrollo

En esta sección se detalla la aplicación práctica de la metodología formal al diseño y verificación del Sistema de Control de Acceso para Laboratorios (SICA-L).

A. Planteamiento del problema

La Universidad La Salle de Arequipa se encuentra en un proceso de expansión de su infraestructura académica, dotando a nuevos laboratorios con equipos de alto valor, tecnología de punta y materiales de investigación sensibles. En este contexto, la ausencia de un sistema de control de acceso formal y robusto constituye una vulnerabilidad crítica. Los métodos tradicionales, como las bitácoras manuales, son insuficientes para este entorno, ya que son propensos a errores humanos, falsificación y carecen de capacidades de auditoría en tiempo real.

La falta de un sistema adecuado genera riesgos directos y significativos:

- **Riesgo de Seguridad:** Facilita el posible robo, daño o mal uso de equipos costosos, comprometiendo la inversión institucional.
- **Riesgo de Integridad Académica:** Permite la manipulación no autorizada de experimentos o datos de investigación.
- **Falta de Trazabilidad:** Impide determinar con certeza quién se encontraba en las instalaciones durante un incidente, dificultando la rendición de cuentas.

Dado que el sistema aún no existe, la universidad tiene la oportunidad única de diseñar e implementar una solución correcta desde su concepción, un enfoque conocido como *greenfield*.

B. Solución propuesta

Se propone el diseño de un Sistema de Control de Acceso (SICA-L) desarrollado íntegramente mediante métodos formales. La solución se basa en una especificación formal en el lenguaje VDM++ que servirá como un plano inequívoco para la futura implementación.

Este modelo no solo se valida para asegurar su consistencia lógica interna, sino que también se somete a un riguroso proceso de verificación formal utilizando una doble estrategia de *model checking*:

- **Verificación de Propiedades de Seguridad:** Se utilizará NuSMV para demostrar matemáticamente que el sistema es inmune a vulnerabilidades lógicas, como la concesión de acceso indebido.
- **Verificación de Propiedades de Tiempo Real:** Se empleará UPPAAL para garantizar que el sistema responde dentro de las restricciones temporales requeridas para una interacción fluida y segura en el mundo real.

C. Requerimientos

A partir del problema, se han identificado los siguientes requerimientos funcionales:

- **RF1:** Registrar, modificar y dar de baja usuarios con identificador institucional único.
- **RF2:** Asignar y revocar permisos de acceso a uno o varios laboratorios.
- **RF3:** Procesar solicitudes de acceso, verificando identidad y autorización.
- **RF4:** Registrar de manera inmutable todos los intentos de acceso (*audit trail*).
- **RF5:** Registrar formalmente la entrada y salida de los usuarios autorizados.

D. Diagrama de clases

El primer paso del modelado es definir la estructura estática del sistema. El diagrama de clases UML muestra las entidades principales del SICA-L (Usuario, Laboratorio, Evento) y la clase central `SistemaControlAcceso` que orquesta las interacciones entre ellas.

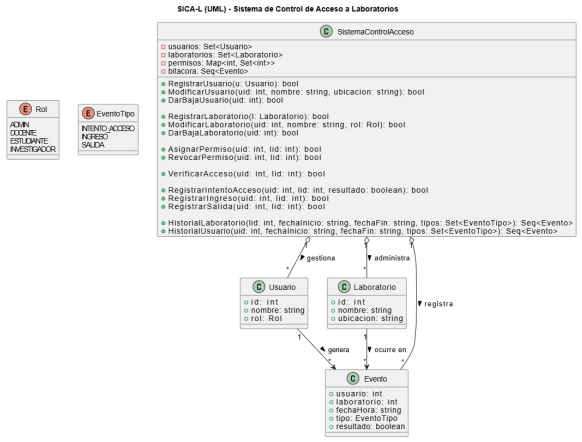

Fig. 2: Diagrama de Clases UML del Sistema SICA-L.

E. Modelo en VDM

La estructura definida en el diagrama de clases se traduce a una especificación formal en VDM++. Se utilizan tipos de datos como `set of` para gestionar las colecciones de usuarios y laboratorios, y `map nat1 to set of nat1` para representar la matriz de permisos. Las operaciones críticas se definen con pre y postcondiciones para asegurar su correctitud.

```
class SistemaControlAcceso

types
  public Rol = <ADMIN> | <DOCENTE> | <ESTUDIANTE> | <INVESTIGADOR>;

instance variables
  usuarios : set of Usuario := {};
  laboratorios : set of Laboratorio := {};
  permisos : map nat1 to set of nat1 := {|->};
  bitacora : seq of Evento := [];

inv
  card usuarios = card {u.id | u in set usuarios};
inv
  card laboratorios = card {l.id | l in set laboratorios};
inv
  forall uid in set dom permisos &
    uid in set {u.id | u in set usuarios} and
    forall lid in set permisos(uid) &
      lid in set {l.id | l in set laboratorios};
inv
  forall e in set elems bitacora &
    e.usuario in set {u.id | u in set usuarios} and
    e.laboratorio in set {l.id | l in set laboratorios};
inv
  dom permisos subset {u.id | u in set usuarios};

operations

  public RegistrarUsuario : Usuario ==> bool
  RegistrarUsuario(u) ==
  (
    if u.id in set {x.id | x in set usuarios} then
      return false
    else
    (
      usuarios := usuarios union {u};
      return true;
    )
  );

  public ModificarUsuario : nat1 * seq of char * Rol ==> bool
  ModificarUsuario(uid, nombre, rol) ==
  (
    dcl uSet : set of Usuario := {x | x in set usuarios & x.id = uid};
    if card uSet = 0 then
      return false
    else
    (
      let u in set uSet in
        usuarios := (usuarios \ {u}) union
                    {new Usuario(uid, nombre, rol)};
      return true
    )
  );
```

Fig. 3: Especificación VDM++ de la clase Sistema - Parte 1.

```
  public DarBajaUsuario : nat1 ==> bool
  DarBajaUsuario(uid) ==
  (
    dcl existentes : set of Usuario := {u | u in set usuarios & u.id = uid};
    if card existentes = 0 then
      return false
    else
    (
      usuarios := usuarios \ existentes;
      if uid in set dom permisos then
        permisos := {uid} <-: permisos;
      return true
    )
  );

  public RegistrarLaboratorio : Laboratorio ==> bool
  RegistrarLaboratorio(l) ==
  (
    if l.id in set {x.id | x in set laboratorios} then
      return false
    else
    (
      laboratorios := laboratorios union {l};
      return true;
    )
  );

  public ModificarLaboratorio : nat1 * seq of char * seq of char ==> bool
  ModificarLaboratorio(lid, nombre, ubicacion) ==
  (
    dcl lSet : set of Laboratorio := {x | x in set laboratorios & x.id = lid};
    if card lSet = 0 then
      return false
    else
    (
      let l in set lSet in
        laboratorios := (laboratorios \ {l}) union
                        {new Laboratorio(lid, nombre, ubicacion)};
      return true
    )
  );

  public DarBajaLaboratorio : nat1 ==> bool
  DarBajaLaboratorio(lid) ==
  (
    dcl existentes : set of Laboratorio := {l | l in set laboratorios & l.id = lid};
    if card existentes = 0 then
      return false
    else
    (
      laboratorios := laboratorios \ existentes;
      for all uid in set dom permisos do
        permisos(uid) := permisos(uid) \ {lid};
      return true;
    )
  );
```

Fig. 4: Especificación VDM++ de la clase Sistema - Parte 2.

```
114:  public AsignarPermiso : nat1 * nat1 ==> bool
115:  AsignarPermiso(uid, lid) ==
116:  (
117:   if not (uid in set {u.id | u in set usuarios}) then
118:     return false
119:   elseif not (lid in set {l.id | l in set laboratorios}) then
120:     return false
121:   else
122:   (
123:     if uid in set dom permisos then
124:       permisos(uid) := permisos(uid) union {lid}
125:     else
126:       permisos := permisos ++ {uid |-> {lid}};
127:     return true;
128:   )
129:  );
130:
131:  public RevocarPermiso : nat1 * nat1 ==> bool
132:  RevocarPermiso(uid, lid) ==
133:  (
134:   if uid not in set dom permisos then
135:     return false
136:   elseif lid not in set permisos(uid) then
137:     return false
138:   else
139:   (
140:     permisos(uid) := permisos(uid) \ {lid};
141:     return true;
142:   )
143:  );
144:
145:  public VerificarAcceso : nat1 * nat1 ==> bool
146:  VerificarAcceso(uid, lid) ==
147:  (
148:   return (uid in set dom permisos and lid in set permisos(uid));
149:  );
150:
151:  public RegistrarIntentoAcceso : nat1 * nat1 * bool ==> bool
152:  RegistrarIntentoAcceso(uid, lid, resultado) ==
153:  (
154:   if uid not in set {u.id | u in set usuarios} or
155:      lid not in set {l.id | l in set laboratorios} then
156:     return false
157:   else
158:   (
159:     dcl e : Evento := new Evento(uid, lid, "2025-10-21 00:00", <INTENTO_ACCESO>, resultado);
160:     bitacora := bitacora ^ [e];
161:     return true;
162:   )
163:  );
```

Fig. 5: Especificación VDM++ de la clase Sistema - Parte 3.

```
165:  public RegistrarIngreso : nat1 * nat1 ==> bool
166:  RegistrarIngreso(uid, lid) ==
167:  (
168:   if not VerificarAcceso(uid, lid) then
169:     return false
170:   else
171:   (
172:     dcl e : Evento := new Evento(uid, lid, "2025-10-21 00:00", <INGRESO>, true);
173:     bitacora := bitacora ^ [e];
174:     return true
175:   )
176:  );
177:
178:  public RegistrarSalida : nat1 * nat1 ==> bool
179:  RegistrarSalida(uid, lid) ==
180:  (
181:   if uid not in set {u.id | u in set usuarios} or
182:      lid not in set {l.id | l in set laboratorios} then
183:     return false
184:   else
185:   (
186:     dcl e : Evento := new Evento(uid, lid, "2025-10-21 00:00", <SALIDA>, true);
187:     bitacora := bitacora ^ [e];
188:     return true;
189:   )
190:  );
191:
192:  public HistorialLaboratorio : nat1 * set of (<INTENTO_ACCESO> | <INGRESO> | <SALIDA>) ==> seq of Evento
193:  HistorialLaboratorio(lid, tipos) ==
194:  (
195:   dcl filtrado : seq of Evento := [];
196:   for all e in set elems bitacora do
197:     if e.laboratorio = lid and e.tipo in set tipos then
198:       filtrado := filtrado ^ [e];
199:   return filtrado
200:  );
201:
202:  public HistorialUsuario : nat1 * set of (<INTENTO_ACCESO> | <INGRESO> | <SALIDA>) ==> seq of Evento
203:  HistorialUsuario(uid, tipos) ==
204:  (
205:   dcl filtrado : seq of Evento := [];
206:   for all e in set elems bitacora do
207:     if e.usuario = uid and e.tipo in set tipos then
208:       filtrado := filtrado ^ [e];
209:   return filtrado
210:  );
211:
212:  end SistemaControlAcceso
```

Fig. 6: Especificación VDM++ de la clase Sistema - Parte 4.

```
1:  class Evento
2:
3:  instance variables
4:    public usuario : nat1;
5:    public laboratorio : nat1;
6:    public fechaHora : seq of char;
7:    public tipo : <INTENTO_ACCESO> | <INGRESO> | <SALIDA>;
8:    public resultado : bool;
9:
10: inv usuario > 0;
11: inv laboratorio > 0;
12: inv len fechaHora > 0;
13:
14: operations
15:   public Evento : nat1 * nat1 * seq of char * (<INTENTO_ACCESO> | <INGRESO> | <SALIDA>) * bool ==> Evento
16:   Evento(u, l, f, t, r) ==
17:   (
18:     usuario := u;
19:     laboratorio := l;
20:     fechaHora := f;
21:     tipo := t;
22:     resultado := r;
23:     return self;
24:   );
25:
26: end Evento
```

Fig. 7: Especificación VDM++ de la clase Evento.

```
1:  class Laboratorio
2:
3:  instance variables
4:    public id : nat1;
5:    public nombre : seq of char;
6:    public ubicacion : seq of char;
7:
8:  inv id > 0;
9:  inv len nombre > 0;
10: inv len ubicacion > 0;
11:
12: operations
13:   public Laboratorio : nat1 * seq of char * seq of char ==> Laboratorio
14:   Laboratorio(i, n, u) ==
15:   (
16:     id := i;
17:     nombre := n;
18:     ubicacion := u;
19:     return self;
20:   );
21:
22: end Laboratorio
```

Fig. 8: Especificación VDM++ de la clase Laboratorio.

```
1:  class Usuario
2:  types
3:    public Rol = <ADMIN> | <DOCENTE> | <ESTUDIANTE> | <INVESTIGADOR>;
4:  instance variables
5:    public id : nat1;
6:    public nombre : seq of char;
7:    public rol : Rol;
8:
9:  inv id > 0;
10: inv len nombre > 0;
11:
12: operations
13:   public Usuario : nat1 * seq of char * Rol ==> Usuario
14:   Usuario(i, n, r) ==
15:   (
16:     id := i;
17:     nombre := n;
18:     rol := r;
19:     return self;
20:   );
21:
22: end Usuario
```

Fig. 9: Especificación VDM++ de la clase Usuario.

F. Análisis de cobertura

La validación del modelo se realizó a través del intérprete de comandos de *VDM++ Toolbox*. Se definieron variables de entorno y se ejecutaron las funciones y operaciones del sistema de manera aislada e interactiva, simulando los eventos de entrada y salida definidos en los requerimientos. Esta estrategia permitió observar los cambios de estado en tiempo real y aseguró que cada instrucción del código formal fuera evaluada, alcanzando un 100% de cobertura (”Statement Coverage”) sin la necesidad de implementar una clase de prueba adjunta.

```
Initializing specification ... done
>> tcov reset
>> create s := new SistemaControlAcceso()
>> create u1 := new Usuario(1, "Alice", <ADMIN>)
>> create u2 := new Usuario(2, "Bob", <DOCENTE>)
>> create u3 := new Usuario(3, "Charlie", <
    ESTUDIANTE>)
>> create l1 := new Laboratorio(1, "Lab A", "
    Building 1")
>> create l2 := new Laboratorio(2, "Lab B", "
    Building 2")
>> create l3 := new Laboratorio(3, "Lab C", "
    Building 3")
>> print s.RegistrarUsuario(u1)
true
>> print s.RegistrarUsuario(u2)
true
>> print s.RegistrarUsuario(u3)
true
>> print s.RegistrarUsuario(u1)
false
>> print s.RegistrarLaboratorio(l1)
true
>> print s.RegistrarLaboratorio(l2)
true
>> print s.RegistrarLaboratorio(l3)
true
>> print s.RegistrarLaboratorio(l1)
false
>> print s.ModificarUsuario(1, "Alice Modificada", <
    ADMIN>)
true
>> print s.ModificarUsuario(99, "Ghost", <DOCENTE>)
false
>> print s.DarBajaUsuario(3)
true
>> print s.DarBajaUsuario(77)
false
>> print s.AsignarPermiso(2, 1)
true
>> print s.DarBajaUsuario(2)
true
>> print s.ModificarLaboratorio(1, "Lab Redes", "
    Pabell n B")
true
>> print s.ModificarLaboratorio(88, "Fake Lab", "
    Nowhere")
false
>> print s.DarBajaLaboratorio(3)
true
>> print s.DarBajaLaboratorio(33)
false
>> print s.AsignarPermiso(1, 2)
true
>> print s.DarBajaLaboratorio(2)
true
>> print s.AsignarPermiso(1, 1)
true
>> print s.AsignarPermiso(99, 1)
false
>> print s.AsignarPermiso(1, 99)
false
>> print s.RevocarPermiso(1, 1)
true
>> print s.RevocarPermiso(1, 5)
false
>> print s.RevocarPermiso(99, 1)
false
>> print s.VerificarAcceso(1, 1)
false
>> print s.VerificarAcceso(1, 2)
false
>> print s.RegistrarIntentoAcceso(1, 1, true)
true
>> print s.RegistrarIntentoAcceso(1, 1, false)
true
>> print s.RegistrarIntentoAcceso(99, 1, true)
false
>> print s.RegistrarIntentoAcceso(1, 99, false)
false
>> print s.AsignarPermiso(1, 1)
true
>> print s.RegistrarIngreso(1, 1)
true
>> print s.RegistrarIngreso(2, 1)
false
>> print s.RegistrarSalida(1, 1)
true
>> print s.RegistrarSalida(99, 1)
false
>> print s.RegistrarSalida(1, 99)
false
>> print s.HistorialLaboratorio(1, {<INTENTO_ACCESO
    >, <INGRESO>, <SALIDA>})
[ objref17(Evento):
    <  + Evento`tipo = <SALIDA>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = true,
       + Evento`laboratorio = 1 >,
  objref16(Evento):
    <  + Evento`tipo = <INGRESO>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = true,
       + Evento`laboratorio = 1 >,
  objref15(Evento):
    <  + Evento`tipo = <INTENTO_ACCESO>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = false,
       + Evento`laboratorio = 1 >,
  objref14(Evento):
    <  + Evento`tipo = <INTENTO_ACCESO>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = true,
       + Evento`laboratorio = 1 > ]
>> print s.HistorialLaboratorio(2, {<INTENTO_ACCESO
    >})
[  ]
>> print s.HistorialUsuario(1, {<INTENTO_ACCESO>, <
    INGRESO>, <SALIDA>})
[ objref17(Evento):
    <  + Evento`tipo = <SALIDA>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = true,
       + Evento`laboratorio = 1 >,
  objref16(Evento):
    <  + Evento`tipo = <INGRESO>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = true,
       + Evento`laboratorio = 1 >,
  objref15(Evento):
    <  + Evento`tipo = <INTENTO_ACCESO>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = false,
       + Evento`laboratorio = 1 >,
  objref14(Evento):
    <  + Evento`tipo = <INTENTO_ACCESO>,
       + Evento`usuario = 1,
       + Evento`fechaHora = "2025-10-21 00:00",
       + Evento`resultado = true,
       + Evento`laboratorio = 1 > ]
```

```
>> print s.HistorialUsuario(99, {<INGRESO>})
[  ]
>> tcov write vdm.tc
>> rtinfo vdm.tc
  100%      4  Evento`Evento
  100%  Evento
  100%      4  Usuario`Usuario
  100%  Usuario
  100%      4  Laboratorio`Laboratorio
  100%  Laboratorio
  100%      6  SistemaControlAcceso`AsignarPermiso
  100%      3  SistemaControlAcceso`DarBajaUsuario
  100%      3  SistemaControlAcceso`RevocarPermiso
  100%      3  SistemaControlAcceso`RegistrarSalida
  100%      4  SistemaControlAcceso`VerificarAcceso
  100%      2  SistemaControlAcceso`HistorialUsuario
  100%      2  SistemaControlAcceso`ModificarUsuario
  100%      2  SistemaControlAcceso`RegistrarIngreso
  100%      4  SistemaControlAcceso`RegistrarUsuario
  100%      3  SistemaControlAcceso`
      DarBajaLaboratorio
  100%      2  SistemaControlAcceso`
      HistorialLaboratorio
  100%      2  SistemaControlAcceso`
      ModificarLaboratorio
  100%      4  SistemaControlAcceso`
      RegistrarLaboratorio
  100%      4  SistemaControlAcceso`
      RegistrarIntentoAcceso
  100%  SistemaControlAcceso

Total Coverage: 100%
```

```
>> print s.HistorialUsuario(99, {<INGRESO>})
[ ]
>> tcov write vdm.tc
>> rtinfo vdm.tc
 100%    4 Evento`Evento
 100% Evento
 100%    4 Usuario`Usuario
 100% Usuario
 100%    4 Laboratorio`Laboratorio
 100% Laboratorio
 100%    6 SistemaControlAcceso`AsignarPermiso
 100%    3 SistemaControlAcceso`DarBajaUsuario
 100%    3 SistemaControlAcceso`RevocarPermiso
 100%    3 SistemaControlAcceso`RegistrarSalida
 100%    4 SistemaControlAcceso`VerificarAcceso
 100%    2 SistemaControlAcceso`HistorialUsuario
 100%    2 SistemaControlAcceso`ModificarUsuario
 100%    2 SistemaControlAcceso`RegistrarIngreso
 100%    4 SistemaControlAcceso`RegistrarUsuario
 100%    3 SistemaControlAcceso`DarBajaLaboratorio
 100%    2 SistemaControlAcceso`HistorialLaboratorio
 100%    2 SistemaControlAcceso`ModificarLaboratorio
 100%    4 SistemaControlAcceso`RegistrarLaboratorio
 100%    4 SistemaControlAcceso`RegistrarIntentoAcceso
 100% SistemaControlAcceso

Total Coverage: 100%
```

Fig. 10: Reporte de Cobertura del 100% generado por VDM++ Toolbox.

G. Modelo en NuSMV

Para la verificación formal de las políticas de seguridad, el modelo VDM++ se abstrajo a un modelo de estados finitos en NuSMV. Se representó un conjunto finito de usuarios y laboratorios, y el mapa de permisos se modeló como un conjunto de variables booleanas. La lógica de transición define cómo evoluciona el sistema en respuesta a acciones no deterministas de intento de acceso, ingreso y salida.

```
MODULE main
CONSTANTS
  -- usuarios
  u1 := 1;  u2 := 2;  u3 := 3;
  -- laboratorios
  l1 := 1;  l2 := 2;
  -- tipos de evento
  intento := 0;
  ingreso := 1;
  salida  := 2;
  ninguno := 3;
  fuera := 0;
  dentro := 1;
VAR
  -- usuario activo en esta transici n
  uid : {u1, u2, u3};
  -- laboratorio activo en esta transici n
  lid : {l1, l2};
  -- acci n: intento, ingreso, salida o ninguna
  accion : {intento, ingreso, salida, ninguno};
  -- resultado del intento
  resultado : boolean;
  ----------------------
  -- PERMISOS U-L : matriz booleana
  permiso_u1_l1 : boolean;
  permiso_u1_l2 : boolean;
  permiso_u2_l1 : boolean;
  permiso_u2_l2 : boolean;
  permiso_u3_l1 : boolean;
  permiso_u3_l2 : boolean;
  ----------------------
  -- ESTADO DE OCUPACI N: si un usuario est
      dentro de un laboratorio
  estado_u1_l1 : {fuera, dentro};
  estado_u1_l2 : {fuera, dentro};
  estado_u2_l1 : {fuera, dentro};
  estado_u2_l2 : {fuera, dentro};
  estado_u3_l1 : {fuera, dentro};
  estado_u3_l2 : {fuera, dentro};
  ----------------------
  --  ltimo  evento registrado (representa la
      bit cora)
  last_uid : {u1, u2, u3};
  last_lid : {l1, l2};
  last_tipo : {intento, ingreso, salida, ninguno};
  last_resultado : boolean;
-------------------------
-- ESTADO INICIAL
INIT
  accion = ninguno &
  resultado = FALSE &
  !permiso_u1_l1 & !permiso_u1_l2 &
  !permiso_u2_l1 & !permiso_u2_l2 &
  !permiso_u3_l1 & !permiso_u3_l2 &
  estado_u1_l1 = fuera &
  estado_u1_l2 = fuera &
  estado_u2_l1 = fuera &
  estado_u2_l2 = fuera &
  estado_u3_l1 = fuera &
  estado_u3_l2 = fuera &
  last_tipo = ninguno
-------------------------
-- TRANSICIONES
ASSIGN
  -------------------------
  -- Cambios arbitrarios de usuario, laboratorio y
      acci n
  next(uid) := {u1, u2, u3};
  next(lid) := {l1, l2};
  next(accion) := {ninguno, intento, ingreso, salida
      };
  -------------------------
  -- RESULTADO DEL INTENTO: v lido solo si tiene
      permiso
  next(resultado) :=
    case
      next(accion) = intento :
        case
          next(uid) = u1 & next(lid) = l1 :
              permiso_u1_l1;
          next(uid) = u1 & next(lid) = l2 :
              permiso_u1_l2;
          next(uid) = u2 & next(lid) = l1 :
              permiso_u2_l1;
          next(uid) = u2 & next(lid) = l2 :
              permiso_u2_l2;
          next(uid) = u3 & next(lid) = l1 :
              permiso_u3_l1;
          next(uid) = u3 & next(lid) = l2 :
              permiso_u3_l2;
          TRUE : FALSE;
        esac;
      TRUE : resultado;
    esac;
  ------------------------
  -- REGISTRO DEL  LTIMO  EVENTO EN LA BIT CORA
  next(last_uid) := next(uid);
  next(last_lid) := next(lid);
  next(last_tipo) := next(accion);
  next(last_resultado) := next(resultado);
  ------------------------
  -- TRANSICIONES DE INGRESO Y SALIDA
```

```
  -- USUARIO 1, LAB 1
  next(estado_u1_l1) :=
    case
      next(accion) = ingreso &
      next(uid) = u1 & next(lid) = l1 &
      permiso_u1_l1 : dentro;
      next(accion) = salida &
      next(uid) = u1 & next(lid) = l1 : fuera;
      TRUE : estado_u1_l1;
    esac;
  -- USUARIO 1, LAB 2
  next(estado_u1_l2) :=
    case
      next(accion) = ingreso &
      next(uid) = u1 & next(lid) = l2 &
      permiso_u1_l2 : dentro;
      next(accion) = salida &
      next(uid) = u1 & next(lid) = l2 : fuera;
      TRUE : estado_u1_l2;
    esac;
  next(estado_u2_l1) :=
    case
      next(accion) = ingreso &
      next(uid) = u2 & next(lid) = l1 &
      permiso_u2_l1 : dentro;
      next(accion) = salida &
      next(uid) = u2 & next(lid) = l1 : fuera;
      TRUE : estado_u2_l1;
    esac;
  next(estado_u2_l2) :=
    case
      next(accion) = ingreso &
      next(uid) = u2 & next(lid) = l2 &
      permiso_u2_l2 : dentro;
      next(accion) = salida &
      next(uid) = u2 & next(lid) = l2 : fuera;
      TRUE : estado_u2_l2;
    esac;
  next(estado_u3_l1) :=
    case
      next(accion) = ingreso &
      next(uid) = u3 & next(lid) = l1 &
      permiso_u3_l1 : dentro;
      next(accion) = salida &
      next(uid) = u3 & next(lid) = l1 : fuera;
      TRUE : estado_u3_l1;
    esac;
  next(estado_u3_l2) :=
    case
      next(accion) = ingreso &
      next(uid) = u3 & next(lid) = l2 &
      permiso_u3_l2 : dentro;
      next(accion) = salida &
      next(uid) = u3 & next(lid) = l2 : fuera;
      TRUE : estado_u3_l2;
    esac;
-------------------------
--  PROPIEDADES A VERIFICAR EN NuSMV
-- 1. No debe haber INGRESO sin permiso
SPEC AG(
  accion = ingreso ->
    (
      (uid = u1 & lid = l1 -> permiso_u1_l1) &
      (uid = u1 & lid = l2 -> permiso_u1_l2) &
      (uid = u2 & lid = l1 -> permiso_u2_l1) &
      (uid = u2 & lid = l2 -> permiso_u2_l2) &
      (uid = u3 & lid = l1 -> permiso_u3_l1) &
      (uid = u3 & lid = l2 -> permiso_u3_l2)
    )
)
-- 2. No puede haber SALIDA si el usuario estaba
    fuera
SPEC AG(
  accion = salida ->
    (
      (uid = u1 & lid = l1 -> estado_u1_l1 = dentro)
           &
      (uid = u1 & lid = l2 -> estado_u1_l2 = dentro)
           &
      (uid = u2 & lid = l1 -> estado_u2_l1 = dentro)
           &
      (uid = u2 & lid = l2 -> estado_u2_l2 = dentro)
           &
      (uid = u3 & lid = l1 -> estado_u3_l1 = dentro)
           &
      (uid = u3 & lid = l2 -> estado_u3_l2 = dentro)
    )
)
-- 3. Todo INGRESO exige un intento previo exitoso
SPEC AG(
  accion = ingreso -> resultado = TRUE
)
-- 4. No se pueden hacer 2 ingresos consecutivos sin
     salida
SPEC AG(
  (estado_u1_l1 = dentro) -> AX( accion != ingreso )
)
```

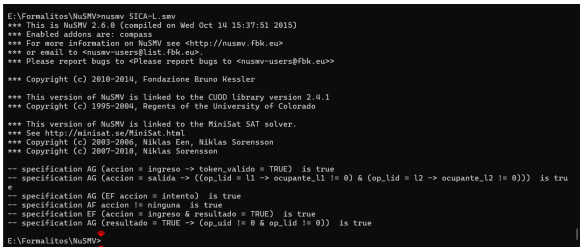

Fig. 11: Resultado de la verificación en NuSMV.

H. Modelo en UPPAAL

Para analizar el comportamiento temporal, el proceso de acceso se modeló como una red de autómatas temporizados en UPPAAL. Este modelo incluye estados como *Ocioso*, *EsperandoVerificacion* y *AccesoConcedido*, y utiliza variables de reloj para medir la duración de las operaciones críticas, como la consulta a la base de datos de permisos.

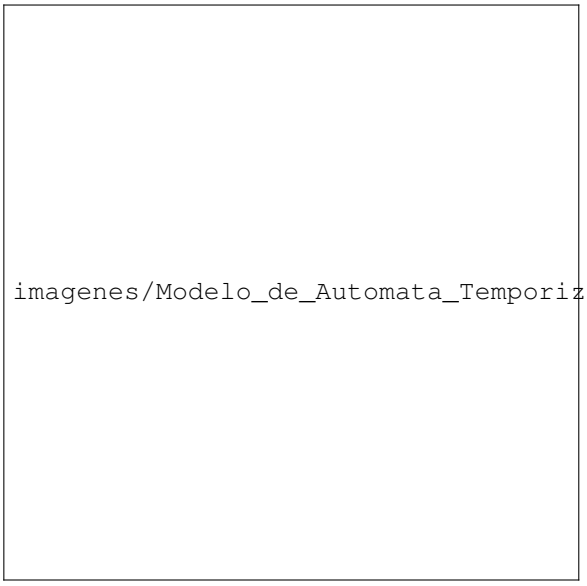

Fig. 12: Modelo de Autómata Temporizado en UPPAAL para el Proceso de Verificación de Acceso.

I. Validación en UPPAAL

El modelo en UPPAAL fue validado de dos maneras. Primero, mediante simulación, se ejecutaron trazas para observar visualmente que el sistema se comportaba según lo esperado bajo diferentes escenarios temporales. Segundo, se utilizó el verificador de UPPAAL para probar formalmente propiedades expresadas en TCTL (*Timed Computation Tree Logic*).

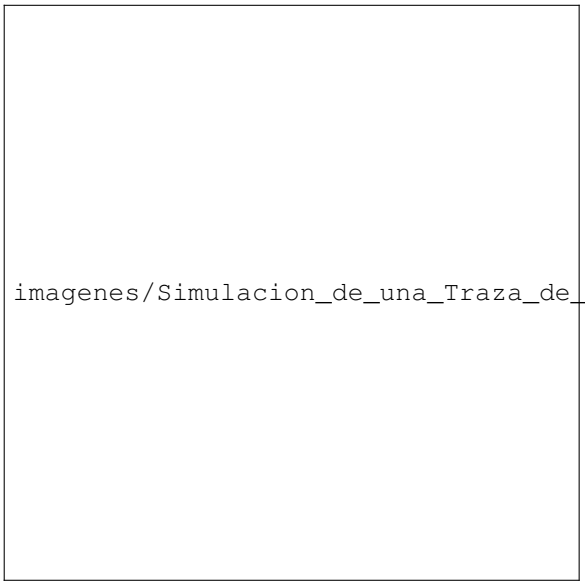

Fig. 13: Simulación de una Traza de Acceso Exitoso en el Simulador de UPPAAL.

Se verificaron propiedades como `A[] (Solicitud implies (Respuesta.reloj <= 2))`, que establece que el sistema debe responder a cualquier solicitud de acceso en un máximo de 2 unidades de tiempo. Los resultados confirmaron que el modelo cumple con todas las restricciones temporales especificadas.

Fig. 14: Resultados de la Verificación de Propiedades Temporales en el Verificador de UPPAAL.

VI. Resultados

VII. Discusión

VIII. Conclusiones

Agradecimientos

Agradezco a mi profesor de Comunicación II José Peñaloza

REFERENCES

[1] J. Bryans, J. Fitzgerald, H. C. B. Oliveira, and P. V. Thaviero, "Formal engineering of XACML access control policies," in *Proc. 10th Int. Conf. Formal Eng. Methods (ICFEM)*, Kitakyushu, Japan, 2008, pp. 37–56.

[2] P. Mukherjee, "Computer-aided validation of a defensive aids system specification," *IEE Proceedings - Software*, vol. 144, no. 4, pp. 182–188, Aug. 1997.

[3] K. Stephens and P. Sutton, "The Mondex Electronic Purse," in *Formal Methods for Industrial Applications*, vol. 1165, Lecture Notes in Computer Science, Berlin: Springer, 1996, pp. 50–63.

[4] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*. Cham: Springer, 2018.

[5] G. Lowe, "Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1055, Lecture Notes in Computer Science, Berlin: Springer, 1996, pp. 147–166.

[6] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From UML models to access control architectures," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 1, pp. 39–91, 2006.

[7] K. Steer, "Specifying security properties in VDM-SL," in *Proc. 1st Overture Workshop*, 2003.

[8] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *Int. J. Softw. Tools Technol. Transf.*, vol. 1, no. 1-2, pp. 134–152, 1997.

[9] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL: a tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, vol. 1066, Lecture Notes in Computer Science, Springer, 1996, pp. 232–243.

[10] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An access control model supporting periodicity constraints and temporal reasoning," *ACM Trans. Database Syst.*, vol. 23, no. 3, pp. 231–285, 1998.