Uso de Especificaciones Formales con VDM en el Acceso de Usuarios en una Red Social Universitaria: Caso Uconecta.

Hernán Chacca, Estudiante[1], Rolfy Montufar, Estudiante[1], and Richard Zenteno, Estudiante[1], Elizabeth Vidal[1], mentor
[1]Universidad Nacional San Agustín, Perú, hchaccac@unsa.edu.pe, rmontufar@unsa.edu.pe, rzentenoc@unsa.edu.pe

Abstract*– Uno de los principales problemas en las redes sociales actuales es la seguridad. Muchas veces el problema es debido a que en los requerimientos a ser modelados no es posible capturar todos los detalles y restricciones los cuales puede llevar a una implementación con riegos de seguridad y ambigüedades. Una de las formas de reducir estos problemas es el uso de especificaciones formales para reducirlos y cubrir todas las restricciones del sistema para hacer que su comportamiento se expresado fehacientemente. Las especificaciones y restricciones serán validadas en su totalidad mediante casos de prueba refinadas con análisis de cobertura. Como ejemplo de aplicación se desarrolla las especificaciones formales del módulo de acceso de usuarios de Uconecta, red social de universitarios. Especificando formalmente esta funcionalidad y posteriormente validándolo se muestra que usando especificaciones formales podemos reducir problemas de seguridad y restricciones en una implementación.

***Keywords*—Especificación formal, Validación, Casos de Prueba, Análisis de Cobertura.**

I. Introducción

En los últimos años internet permitió un gran incremento de comunidades virtuales o redes sociales [1]. Una de las fases críticas del desarrollo de todo sistema como estas redes sociales es la identificación y especificación de los requerimientos [2]. Existen métodos que nos permiten las especificaciones y restricciones más refinadas conforme a los requerimientos, como UML [3, 4], que a pesar que es un lenguaje de modelamiento estándar, no se pueden obtener el refinamiento de especificaciones completa y siempre existe ambigüedades [5]. Para complementar métodos de modelamiento y especificación como UML (Unified Modeling Language), se desarrollaron métodos formales que nos permiten notaciones más precisas además de la aplicación de casos de prueba para la validación de las especificaciones [6].

En la Universidad Nacional San Agustín en el año 2016 se desarrolló Uconecta, una red social universitaria. Uno de los temas críticos en esta red es la seguridad, en especial la seguridad en el acceso de usuarios. Una forma de reducir riesgos, ambigüedades y no conformidades en la implementación es la validación formal de las especificaciones. Por ello nuestro trabajo presenta el desarrollo de las especificaciones formales del módulo de acceso de usuarios en Uconecta a partir de un modelo conceptual. Para conseguir este objetivo, las especificaciones formales se desarrollaron en VDM++ (Vienna Development Method++), un lenguaje de especificación formal. Se utilizó VDM++ToolBox como herramienta para validar las especificaciones con análisis de cobertura.

El principal aporte de este trabajo es mostrar que la aplicación de especificaciones formales contribuye a refinar especificaciones y reducir ambigüedades a la hora de implementar los requerimientos, por lo tanto reducir riesgos de seguridad y el cumplimiento incompleto de restricciones que podrían presentarse en un sistema. Mostramos como caso de aplicación Uconecta y los riesgos que podrían presentar en el módulo de acceso de usuarios.

El resto del artículo está organizado de la siguiente manera: En la sección 2 se presentan trabajos relacionados en donde vemos algunas aplicaciones de especificaciones formales con diferentes enfoques. En la sección 3 se describe los fundamentos en los que hemos basado el presente trabajo y las características de VDM++ que se utilizó en el desarrollo del proyecto agregando una breve descripción de la herramienta utilizada para validar nuestra propuesta. En la sección 4 se presenta como caso de aplicación a Uconecta, se describen los requerimientos de manera informal para luego especificarlos de manera formal con VDM++. También se presenta el análisis y la validación de las especificaciones utilizando la herramienta VDM++ToolBox. Finalmente presentamos nuestras conclusiones en la sección 5.

II. Trabajos Relacionados

El tema de uso de especificaciones formales para seguridad para accesos a sistemas ha sido tratado en varias investigaciones, así por ejemplo en el trabajo de Ledru [7] se discute el uso de técnicas de pruebas y simulación para validar políticas de control de acceso expresado en SecureUML. Mediante un caso de estudio de un sistema de información de seguridad médica y utilizando el modelo SecureUML se expresa los modelos de información y modelos de seguridad. Esto es traducido a un lenguaje de especificación formal, en este caso el lenguaje B para validar las modelos funcionales y actividades que incluyen escenarios basados en casos de uso, realizar pruebas sistemáticas de permisos de roles e investigar posibles ataques.

Digital Object Identifier: (to be inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).

Muhammad en [8] utiliza especificaciones formales enfocadas en aspectos de seguridad de un sistema de software. Se describe propiedades de seguridad desarrolladas en VDM para especificar propiedades completas, sin ambigüedades y precisos. Se hizo *type checking*, *sintaxt checking* y análisis de los resultados utilizando la herramienta VDM-SL (Vienna Development Method – Specification Language) ToolBox. Se especifican propiedades como Autorización, Autenticación, Integridad, Confidencialidad y disponibilidad de recursos. Se presenta primero los modelos de manera informal, luego describen la especificación formal para cada propiedad de seguridad.

Azeem en [9] desarrolla especificaciones para una aplicación E-Healt System usando un lenguaje y esquema de especificacion formal Z. Hace notar que las especificaciones formales no solo son para mejorar la confiabilidad, correctitud y completitud de sistemas criticos, sino tambien para aplicaciones orientadas al comercion y el negocio ya que el tiempo de desarrollo, verificacion y mantenimiento se reducen drasticamente.

En el presente trabajo, evaluamos los requerimientos de autenticacion y validacion de usuarios de la red social universitaria Uconecta [10] utilizando especificaciones con VDM++ a partir de un modelo conceptual en UML de un sistema real como caso de aplicación.

III. Especificaciones Formales y VDM++

En esta sección presentamos los conceptos relevantes que nos permitirán desarrollar nuestro trabajo de tal forma que cada aspecto tratado sea entendido.

A. *Especificaciones Formales*

Según [8] los métodos formales están basados en conceptos matemáticos para la especificación, desarrollo verificación de sistemas de software y hardware en sus diferentes etapas de desarrollo. Especificaciones formales es una categoría específica de métodos formales y se centra en una de las etapas más críticas del desarrollo de software, que es la especificación de requerimientos.

Los requerimientos iniciales de un usuario generalmente son descritos de manera informal, por lo que no es posible probar que todos los requerimientos del usuario hayan sido satisfechos. Al utilizar métodos formales nos solo se consigue que las especificaciones pierdan ambigüedades, sino también se reduce la complejidad del modelo conceptual, la correctitud y consistencia de las especificaciones pueden ser verificadas y revelar algunos errores en una fase temprana para no tener un impacto significativo en fases posteriores como la implementación [11].

Para describir las especificaciones de manera correcta y sin ambigüedades se debe utilizar alguna notación especial, usualmente un lenguaje de especificación formal completo como VDM++, ASTRAL, Z, Object Z, LARCH, RAISE u otras más descritas, analizadas y comparadas en [11, 12].

B. *VDM++*

En este trabajo utilizaremos VDM++ como lenguaje de especificación formal ya que nos permite especificar y validar sistemas orientados a objetos con comportamientos en tiempo real y paralelos [13]. El lenguaje está basado en VDM-SL [14] y desarrolla conceptos de clases, objetos y herencia para especificar las invocaciones de las secuencias de métodos permitidos [15]. En este trabajo describiremos algunos aspectos del lenguaje que serán meramente necesarios para describir el Caso de Aplicación.

C. *Clases*

Los modelos de VDM++ están representados como clases. La clase se representa con la palabra reservada *class*, seguida por el nombre de la clase. Como se aprecia en la Fig. 1, en la estructura de una clase se puede identificar bloques precedidos por palabras reservadas que indican el tipo de elemento que se describen en dicho bloque: *a) Instance variables*, que modelan el estado interno de una clase. *b) Types*, como tipos de datos. *c) Values*, constantes. *d) Operations*, operaciones que una clases puede tener [15].

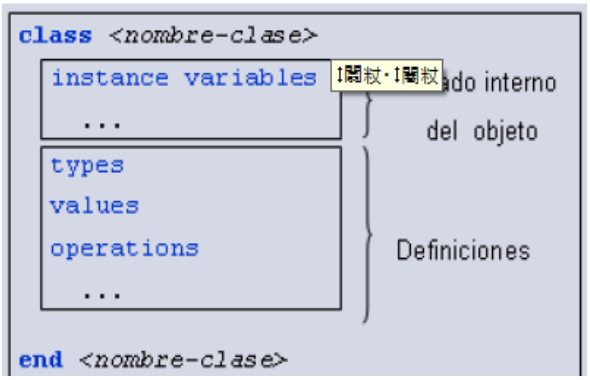

Fig. 1 Estructura de clase en VDM++.

D. *Tipos*

Con VDM++ podemos trabajar con tipos de datos básicos y compuestos. Tipos básicos como booleano (*bool*), numérico (*num*), caracteres (*char*), *quote*, y un tipo *token*. Todas los tipos cuentan con operaciones genéricas como igualdad y desigualdad, y algunas que cuentan con algunas operaciones más como operadores lógicos para tipos booleanos y operaciones básicas matemáticas para tipos numéricos [15]. Como tipos compuestos, para este trabajo, utilizamos Conjuntos (*Set types*) por permitir elementos únicos por conjunto.

Algunas operaciones de Conjuntos que utilizaremos y que soporta el lenguaje son: Pertenencia (*Membership y Not Membership*), que indica con un valor booleano si una elemento está presente en un conjunto y Unión (*Union*) que retorna un nuevo conjunto de la unión de dos conjuntos. Se puede observar la descripción de cada operación en la TABLA I.

TABLA I
DESCRIPCIÓN DE MÉTODOS PARA TIPO "SET"

Operator	Name	Type
e in set s1	Membership	A * set of A → bool
e not in set s1	Not membership	A * set of A → bool
s1 union s2	Union	set of A * set of A → set of A

E. Invariante

Si una variable contiene valores que no serán permitidos, entonces es posible poner una restricción a dichos valores por medio de invariantes. El resultado de dicha será que el tipo será restringido a un conjunto de sus valores originales. La invariante se representa con la palabra reservada "*inv*" que va seguido de una expresión que mientras sea verdadera mantiene en sus límites al valor a restringir [15].

F. Pre Post Condiciones

Las funciones y operaciones pueden ser definidas explícitamente por medio de la definición explícita del algoritmo o implícitamente por medio de las pre/post condiciones. Una precondición es una expresión que debe mantener algunas restricciones antes de que la operación o función sea evaluada. Un post condición en contraste es una expresión que mantienen algunas restricciones luego de que la operación o función sea evaluada [15].

G. Funciones y Operaciones

En VDM++ los algoritmos pueden ser definidos mediante Funciones y Operaciones. Los que diferencia funciones de operaciones es el uso de variables locales y globales. Las Operaciones pueden manipular tanto variables locales como globales. Funciones en cambio no pueden acceder a variable globales y no se les permite definir variable locales. Las funciones son puramente aplicativos mientras que las operaciones son imperativos [15].

H. Expresiones Cuantificadoras

Las expresiones cuantificadoras son un tipo de expresión lógica usadas frecuentemente cuando es necesario hacer una aserción a cerca de una colección de valores. Existen dos tipos de expresiones cuantificadoras: Cuantificadores universales (*forall*) y cuantificadores existenciales (*exists*). Cada uno de ellos enlaza una o más variables a un valor de un conjunto o una lista y los evalúa contra un valor booleano [15, 16].

I. Herramienta

VDM++ToolBox [17] es una herramienta que soporta análisis de modelos formales mediante análisis sintáctico, análisis de tipos, generación de código y análisis de cobertura (Fig. 2). Una de la funcionalidad que utilizaremos en este proyecto es convertir modelos y diagramas de clases UML creados en Rational Rose a especificaciones VDM++.

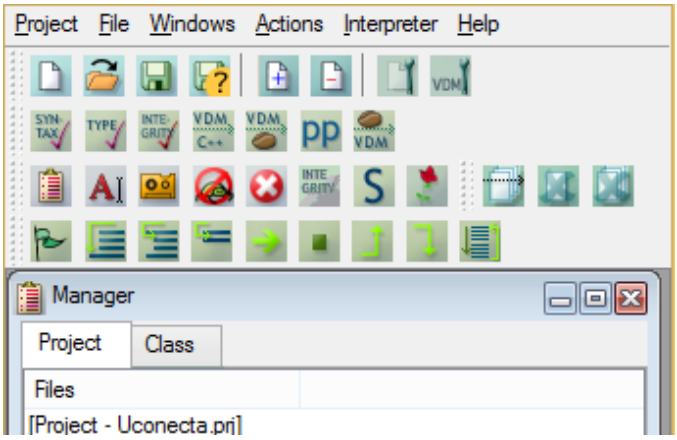

Fig. 2 Herramienta VDM++ ToolBox

Después de plasmar especificaciones en diagramas de clases UML, se tiene que verificar la correctitud de las clases. El analizador sintáctico verifica la ausencia de errores sintácticos de acuerdo a la sintaxis de VDM++. Si la sintaxis de la especificación es correcta, continuamos con el análisis de tipos también de acuerdo a las reglas o tipos nativos de VDM++. Después de hacer las verificaciones respectivas, la herramienta provee soporte para la validación de pruebas utilizando un intérprete. El intérprete permite hacer pruebas por líneas de comando o pruebas en lote mediante scripts.

IV. CASO UCONECTA

A. Descripción General

Uconecta [10] es una red social universitaria desarrollada por estudiantes universitarios con el objetivo de conectar universitarios de diferentes carreras y universidades. El enfoque que toma esta red social es la educación colaborativa por lo que la principal actividad dentro de este sistema es compartir experiencias y conocimiento sobre un curso en una carrera o universidad que puede servir para un estudiante en un centro de estudio diferente generando así un ambiente de aprendizaje colaborativo.

El proyecto Uconecta inicio en Abril del 2015 en la Escuela Profesional Ingeniería de Sistemas [18] de la Universidad Nacional San Agustín [19]. Después de pasar por un periodo de desarrollo y lanzamiento oficial, se hizo pruebas piloto empezando con alumnos de algunos cursos. Actualmente Uconecta es un medio de comunicación y colaboración importante en toda nuestra escuela.

El proyecto está desarrollado sobre la plataforma Drupal [20]. La parte front-end está desarrollado con HTML, JavaScript y CSS3. A través de servicios web se conecta al back-end que está desarrollado sobre el núcleo de Drupal.

Para el registro e ingreso de usuarios se utilizó la API de Facebook por lo tanto, para ingresar a Uconecta es necesario tener una cuenta Facebook (Fig. 3)

Fig. 3 Página principal y de acceso de Uconecta.

Dentro de la red social un usuario tiene la posibilidad de unirse a una Universidad, a una carrera y posteriormente enrolarse a uno o varios cursos (Fig. 4). Para publicar experiencias o conocimiento se tienen que especificar el curso, haciendo del material compartido accesible solo para usuarios conectados mediante el curso (Fig. 5).

Fig. 4 Selección de universidad y cursos.

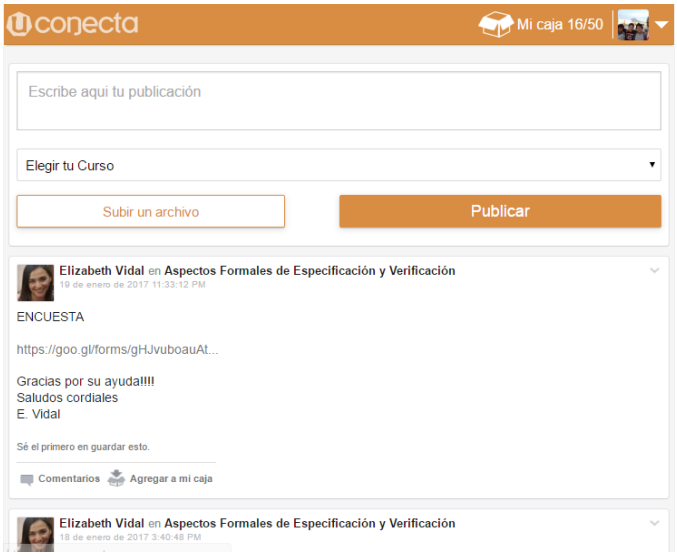

Fig. 5 Noticias de Uconecta.

B. *Problemática*

La autenticación y validación de usuarios es a través de un servicio externo a esta red social, utilizando el API de Facebook. Uconecta busca que sus usuarios sean validados y verificados para que puedan interactuar con esta red social. Después de que un usuario ingrese a Uconecta por Facebook, necesita verificar su cuenta ingresando un código único de estudiante que provee la universidad a cada estudiante como muestra en la Fig. 6.

Fig. 6 Validación de usuario mediante código único de estudiante.

Dado que la parte del ingreso de un usuario no fue validado formalmente, el objetivo de este trabajo es desarrollar una representación del módulo de inicio y registro de usuarios como una versión especificada con un lenguaje de especificación formal.

Siguiendo la metodología desarrollada en [6] enumeramos algunas restricciones como requerimientos para controlar el acceso de los usuarios a recursos presentes en Uconecta.

R1. Un usuario se registra mediante Facebook.

R2. El usuario tiene que escoger una universidad.

R3. Luego de R2 el usuario selecciona su carrera profesional que debe pertenecer a la universidad seleccionada.

R4. Luego de R3 usuario selecciona unirse a uno o varios cursos que estén disponibles en la carrera profesional seleccionada.

R5. El sistema tiene un servicio para verificar si un estudiante pertenece a una universidad, por lo que el usuario debe ingresar el código único de estudiante para validar su cuenta, y esto lo hace después de R4.

R6. Un usuario sin verificar está en un estado desactivado y activado sin esta verificado.

C. *Diagrama de Clases*

El modelo conceptual se puede representar mediante el uso de técnicas orientadas a objetos, por ejemplo con el diagrama de clases, atributos y métodos de una entidad [3]. Representaremos solo la funcionalidad que trataremos en este trabajo, la validación de usuarios. Aprovechando la característica de la herramienta VDM++ToolBox que nos permite pasar diagramas UML [4] a especificación formal VDM++, utilizaremos Rational Rose [21] para representar el modelo conceptual mediante diagrama de clases (Fig. 7).

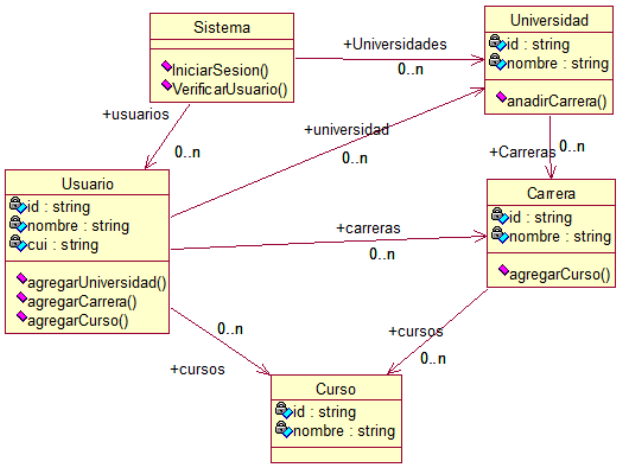

Fig. 7 Modelo conceptual de Uconecta

D. *Especificación Formal*

A partir del modelo conceptual, a través de la herramienta VDM++ToolBox pasamos nuestro modelo conceptual a especificaciones formales. La especificación se hizo en VDM++ y nos centraremos en una clase específica donde se realiza la parte de inicio de sesión de un usuario. La Fig. 8, Fig. 9 y Fig. 10 muestran la especificación de la clase *Sistema* y algunas operaciones que cubren todo el proceso que un usuario sigue para entrar a Uconecta.

El sistema debe tener usuarios y universidades de forma única, no pueden existir dos instancias de estos en el mismo sistema. De la misma forma las universidades tienen Carreas únicas, las carreras tienen cursos únicos, los usuarios están matriculados en carreras y cursos únicos. Como se muestra en las Fig. 8, Fig. 9 y Fig. 10, el tipo compuesto en VDM++ que asegura la no repetición de instancias de un tipo en una lista y el cual utilizaremos en la especificación formal de este sistema es el tipo Conjunto (*Set*).

Para evitar un documente extenso, no se muestra las especificaciones de las clases como *Usuario, Curso, Carrera y Universidad* y nos enfocaremos en el proceso de ingreso del usuario asumiendo que los métodos de las clases entes mencionadas que vamos a utilizar ya están especificadas.

Para R1 se debe ingresar al sistema mediante una cuenta Facebook, por lo cual Uconecta concede el acceso a un usuario ya sea nuevo o uno ya registrado. Un usuario nuevo se guarda en el conjunto de usuarios del sistema y un usuario ya registrado simplemente obtiene el acceso. El método *IniciarSesion* de la línea 22 de la Fig. 8 especifica este requerimiento.

En R2 el usuario debe seleccionar una universidad para ellos agregamos una universidad o varias al conjunto de universidades que el usuario tiene registrado. Para esto el método *SeleccionarUniversidad* es implementado formalmente en la línea 33 de la Fig. 8.

```
1: --
2: -- THIS FILE IS AUTOMATICALLY GENERATED!!
3: --
4: -- Generated at 15-Dec-16 by the UML-VDM++ Link
5: --
6: class Sistema
7: types
8:   public string = seq of char;
9:
10: instance variables
11:   public usuarios : set of Usuario := {};
12:   public Universidades : set of Universidad := {};
13:   public cuis: set of Usuario`codigo;
14:
15:   inv forall us in set usuarios &
16:         us.GetEstado() = <desactivado> or
17:          (us.GetEstado() = <activado> and
18:           us.GetCui() in set cuis
19:          );
20:
21: operations
22:   public IniciarSesion : Usuario ==> bool
23:   IniciarSesion(usuario) ==
24:    if usuario in set usuarios then
25:     return true
26:    else
27:     (
28:     usuarios := usuarios union {usuario};
29:     return true
30:     )
31:   post usuario in set usuarios;
32:
33:   public seleccionarUniversidad: Usuario*set of Universidad ==> Sistema
34:   seleccionarUniversidad(usuario, setUniversidades) ==
35:    (
36:    for all uni in set setUniversidades do usuario.agregarUniversidad(uni);
37:    return self;
38:    )-
```

Fig. 8 Especificación formal parte 1

En R3 el usuario debe seleccionar una o varias carreras así como indica el método en la línea 40 de la Fig. 9. Las carreras elegibles tienen que pertenecer a las universidades seleccionadas previamente caso contrario se presentaría un inconsistencia, y esta consideración se representa mediante una precondición del método definida en la línea 46 de la Fig. 9. Al finalizar el usuario debe tener registrado las carreras seleccionadas por lo que esta condición es asegurada por una postcondicion en la línea 50 de la Fig. 9.

En R4 el usuario debe seleccionar uno o varios cursos, como indica el método *seleccionarCursos* de la línea 52 en la Fig. 9. Del mismo modo, solo son elegible cursos que pertenecen a las carreras seleccionadas por el usuario previamente y la precondición en la línea 58 de la Fig. 9 asegura esta condición. A terminar el método el usuario debe tener registrado todos los cursos seleccionados y la postcondicion en la línea 62 de la Fig. 9 asegura esta condición.

```
40: public seleccionarCarreras: Usuario*set of Carrera ==> Sistema
41: seleccionarCarreras(usuario, carrs) ==
42:  (
43:  for all car in set carrs do usuario.agregarCarrera(car);
44:  return self;
45:  )
46: pre usuario.universidad <> {} and
47:    forall c in set carrs &
48:      exists u in set usuario.universidad &
49:        c in set u.Carreras
50: post forall c in set carrs & c in set usuario.carreras;
51:
52: public seleccionarCursos: Usuario*set of Curso ==> Sistema
53: seleccionarCursos(usuario, cursos) ==
54:  (
55:  for all curso in set cursos do usuario.agregarCurso(curso);
56:  return self;
57:  )
58: pre usuario.carreras <> {} and
59:    forall c in set cursos &
60:      exists ca in set usuario.carreras &
61:        c in set ca.cursos
62: post forall c in set cursos & c in set usuario.cursos;
```

Fig. 9 Especificación formal parte 2

En R5 se debe verificar que el usuario pertenece a alguna de las universidades registradas en el sistema mediante. Para ellos el sistema cuenta con un conjunto de código inicializados en línea 13 de la Fig. 8. Si el código de un usuario se encuentra en el conjunto de códigos del sistema, el usuario cambia su estado a "*activado'*. El método encargado de cubrir este requerimiento se especifica en la línea 64 de la Fig. 10. Para verificar obviamente se tiene que tener al menos un código registrado en el sistema.

```
63:
64: public VerificarUsuario : Usuario ==> bool
65: VerificarUsuario(usuario) ==
66:  if usuario.GetCui() in set cuis then
67:    (usuario.SetEstado(<activado>);
68:    return true)
69:  else
70:    return false
71: pre cuis <> {};
72:
73:
74: public agregarUniversidad: Universidad ==> ()
75: agregarUniversidad(uni) ==
76:  (
77:    Universidades := Universidades union {uni};
78:  );
79:
80:
81: end Sistema
```

Fig. 10 Especificación formal parte 3

Un usuario tiene dos estados según R6 por lo que los cui de los usuarios activados deben estar registrados en el conjunto de cuis del sistema, a menos que aun el usuario aun no este activado, y dicha condición los representamos en una invariante que inicia en la línea 15 de la Fig. 8.

E. Validación

La validación atraves de los casos de pruebas incrementa la confiabilidad del sistema por lo cual se realizo casos de prueba que abarque la mayor parte del codigo del sistema modelado formalmente. A este enfoque se denomina Analisis de Covertura y para lo cual desarrollamos casos de prueba en Test1 especificado en Fig. 11 y Fig. 12.

En la Fig. 13 y Fig. 14 se muestra el resultado de caso de prueba desarrollado y como se puede observar el Analisis de Cobertura esta en un 100% como debe de ser para una validacion correcta y confiable.

```
class Test1

instance variables
  u1: Usuario;
  u2: Usuario;
  cu1: Curso;
  cu2: Curso;
  ca1: Carrera;
  ca2: Carrera;
  uni1: Universidad;
  uni2: Universidad;
  s1: Sistema;
  log: seq of char := [];
operations

public run : () ==> ()
run() == test1();

public test1 : () ==> ()
test1() ==
  (
    u1:= new Usuario("12", "hernan", 20133320);
    logger("nuevo usuario: "^u1.GetId()^":"^u1.GetNombre());

    u2:= new Usuario("11", "Faustino", 20132545);
    logger("nuevo usuario: "^u2.GetId()^":"^u2.GetNombre());

    uni1 := new Universidad("12", "Universidad Nacional San Agustin");
    logger("nuevo universidad: "^uni1.GetId()^":"^uni1.GetNombre());

    ca1 := new Carrera("12", "Ingenieria de Sistemas");
    logger("nuevo carrera: "^ca1.GetId()^":"^ca1.GetNombre());

    cu1 := new Curso("122", "Estructura de Datos");
    logger("nuevo curso: "^cu1.GetId()^":"^cu1.GetNombre());

    cu2 := new Curso("123", "Fundamentos de Programacion");
    logger("nuevo curso: "^cu2.GetId()^":"^cu2.GetNombre());
```

Fig. 11 Caso de Prueba Test1 parte 1

```

    ca1.agregarCurso(cu1);
    ca1.agregarCurso(cu2);

    uni1.anadirCarrera(ca1);

    s1 := new Sistema();
    s1.cuis := {20133320, 20132545, 20131886, 20145896};
    s1.agregarUniversidad(uni1);

    if s1.IniciarSesion(u1) then
      if s1.seleccionarUniversidad(u1, {uni1})
          .seleccionarCarreras(u1, uni1.Carreras)
          .seleccionarCursos(u1, ca1.cursos).VerificarUsuario(u1) then
        logger("asd");

    if s1.IniciarSesion(u1) then
      logger("nuevo inicio sesion");

  );
public logger : Sistema`string ==> ()
logger(str) ==
  (
    log:=log^str;
  );

end Test1
```

Fig. 12 Caso de Prueba Test1 parte 2

```
>> rtinfo vdm.tc
 100%     8 Curso`Curso
 100%     8 Curso`GetId
 100%     8 Curso`SetId
 100%     8 Curso`GetNombre
 100%     8 Curso`SetNombre
 100%     1 Test1`run
 100%     1 Test1`test1
 100%     8 Test1`logger
 100%     4 Carrera`GetId
 100%     4 Carrera`SetId
 100%     4 Carrera`Carrera
 100%     4 Carrera`GetNombre
 100%     4 Carrera`SetNombre
 100%     8 Carrera`agregarCurso
 100%     6 Sistema`IniciarSesion
 100%     4 Sistema`VerificarUsuario
 100%     4 Sistema`seleccionarCursos
 100%     4 Sistema`agregarUniversidad
 100%     4 Sistema`seleccionarCarreras
 100%     4 Sistema`seleccionarUniversidad
```

Fig. 13 Resultado Analisis de Cobertura parte 1

```
 100%     8 Usuario`GetId
 100%     8 Usuario`SetId
 100%     4 Usuario`GetCui
 100%     8 Usuario`SetCui
 100%     8 Usuario`Usuario
 100%     4 Usuario`GetEstado
 100%     8 Usuario`GetNombre
 100%     2 Usuario`SetEstado
 100%     8 Usuario`SetNombre
 100%     8 Usuario`agregarCurso
 100%     4 Usuario`agregarCarrera
 100%     4 Usuario`agregarUniversidad
 100%     4 Universidad`GetId
 100%     4 Universidad`SetId
 100%     4 Universidad`GetNombre
 100%     4 Universidad`SetNombre
 100%     4 Universidad`Universidad
 100%     4 Universidad`anadirCarrera

Total Coverage: 100%
```

Fig. 14 Resultado Analisis de Cobertura parte 2

V. Conclusión

En este trabajo se ha presentado las especificaciones formales del módulo de ingreso de usuarios en Uconecta a partir de un modelo conceptual. El usar especificaciones formales nos permitió reducir las ambigüedades que podrían causar problemas de seguridad en la administración del ingreso de usuarios. Las especificaciones formales se desarrollaron un el lenguaje de especificación formal, VDM++. En conjunto con la herramienta que soporta este lenguaje, VDM++ ToolBox, se consiguió alcanzar el 100% en el análisis de cobertura, lo cual indica la validación de las

especificaciones sin ambigüedades y el incremento de la conformidad entre los requerimientos y la implementación de la misma.

ACKNOWLEDGMENT

Los autores agradecemos a David Jeyachandran por permitirnos ser parte del proyecto Uconecta y Elizabeth Vidal que nos alentó a escribir este trabajo sobre nuestra experiencia.

REFERENCIAS

[1] H. Thomas, "Learning spaces, learning environments and the displacement of learning", British Journal of Educational Technology, 41(3),502-511, 2010

[2] Sommerville, "Software Engineering", Sixth Edition, Addison Wesley, 2001

[3] Z. Liu, "Object oriented software sevelopment with UML", de International Institute for Software Technology, 2002.

[4] "Unified Modeling Language". En línea. Disponible en http://www.uml.org/. Último acceso: 12 de Diciembre de 2016.

[5] B. K. Aicherning, "Systematic Black-Box Testin of Computer-Based System through Formal Abstraction Techiniques". Phd Thesis. Technischen Universit¨at Graz, Germany, 2001.

[6] CSK System Corporation, VDM++ Method Guidelines, 2009.

[7] I. Ledru, A. Idani y J.-L. Richier, "Validation of a security policy by the test of its formal B specification - a case study", Proceedings of the Third FME Workshop on Formal Methods in Software Engineering, 2015.

[8] H. Muhammad Tahir, M. Nadeen, A. Shouket, Z. Raza, S. Hussain y N. A. Zafar, "Formalization of security properties using VDM-SL", International Conference on Information and Communication Technologies (ICICT), 2015.

[9] M. W. Azeem, M. Ahsan, N. M. Minhas y K. Noreen, "Specification of e-health system using Z: A motivation to formal methods", International Conference for Convergence of Technology (I2CT), 2014.

[10] "Uconecta", En Línea. Disponible en https://www.uconecta.com. Ultimo acceso: 12 de diciembre de 2016.

[11] V. B. Misic y D. M. Velasevic, "Formal specifications in software development: An overview", Yugoslav Journal of Operations Research, 1997.

[12] R. Sammi, I. Rubab y M. A. Qureshi, "Formal specification languages for real-time systems", 2010 International Symposium on Information Technology, 2010.

[13] E. Durr y J. V. katwijk, "A Formal Specification Language For Object-Oriented Designs", Proceedings 6th Annual European Computer Conference, Compeuro, 1992.

[14] J. Fitzgerald and P.Gorm Larsen, "Modelling Systems Practical Tools and Techniques in Software Development", 2ª Ed, Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998.

[15] CSK Systems Corporation, "The VDM++ Language Manual", 2009.

[16] Fitzgerald, J., Larsen, P. G., Mukherjee, P., Plat, N., & Verhoef, M. "Validated Designs for Object-Oriented Systems", Springer Science & Business Media, 2005, pp. 99.

[17] CSK Systems Corporation, "VDMTools User Manual (VDM++), 2009.

[18] "Escuela Profesional Ingeniería de Sistemas", En Línea. Disponible en http://www.episunsa.edu.pe/web. Ultimo acceso: 12 de diciembre de 2016.

[19] "Universidad Nacional San Agustin", En Línea. Disponible en http://www.unsa.edu.pe/. Ultimo acceso: 12 de diciembre de 2016

[20] "Drupal", En línea. Disponible en https://www.drupal.org/. Ultimo acceso: 20 de Enero de 2017.

[21] IMB, "IBM Rational Rose". En línea. Disponible en http://www-03.ibm.com/software/products/es/enterprise. Ultimo acceso: 20 de Enero de 2017.