Reduciendo la brecha entre los requerimientos y el código: Overture o VDMToolBox? Análisis comparativo de herramientas para generación de código desde Especificaciones Formales

Ángel Luna Calderón, Harold Cáceres Zea, Jerson Herrera Rivera
Universidad Nacional de San Agustín Arequipa, Perú, angel13.18.15@gmail.com, harold.caceres.zea@gmail.com, jerherrerar@gmail.com

Abstract.-

Hoy en día la necesidad de software confiable en la industria y en la sociedad se ha hecho más evidente. Mas aun si estos son de carácter crítico. Ante estos desafíos los ingenieros de software utilizan técnicas basadas en matemáticas como las especificaciones formales. EL uso de especificaciones formales para que sea costo-efectiva debe ser asistido por algún tipo de herramienta que permita no sólo la validación de especificaciones sinó además la generación de código. En este artículo presentamos el análisis comparativo de dos herramientas que permiten realizar especificaciones formales en el lenguaje VDM++ y generar código Java. Nuestra principal contribución es presentar una ayuda para los desarrolladores de software que se inician en el uso de especificaciones formales para que puedan reutilizar código previamente validado.

I. I. INTRODUCCION

.

Las especificaciones formales son un lenguaje basado en matemáticas cuyo propósito en asistir en la construcción de sistemas de software El tiempo que se invierte en la especificación y el diseño se recupera al tener un producto d alta calidad. El uso de especificaciones formales implica invertir mayor esfuerzo en las etapas tempranas del desarrollo de software. Su uso reduce los errores por ambigüedad en los requerimientos y obliga a un análisis detallado de requerimientos. [1].

Requerimientos incompletos e inconsistencias son puestos al descubierto y resueltos Asistidas generalmente por una herramienta permiten describir los requerimientos de un sistema, analizar su comportamiento, verificar propiedades de interés y muchas veces generar código. El poder tener código generado desde una especificación formal no sólo nos permite tener requerimientos menos ambiguos sino tener un código que reproduzca dichos requerimientos.. En este artículo presentamos el análisis comparativo de dos herramientas que permiten realizar especificaciones formales en el lenguaje VDM++ y generar código Java : VDMToolBox y Overture.

El resto del artículo está organizado de la siguiente manera. En la sección 2 damos una breve descripción del concepto de especificación formal. En la sección 3 presentamos las características de VDMToolBox. En la sección 4 presentamos las características de Overture. En la sección 5 mediante utilizamos la descripción de un proyecto realizado en nuestra universidad realizamos el análisis comparativo de ambas herramientas. En la sección 6 presentamos los resultados obtenidos. Finalmente presentamos nuestras conclusiones.

II. ESPECIFICACION FORMAL

A. Definición

La especificación formal o métodos formales son: técnicas, lenguajes y herramientas definidos matemáticamente, para especificar y verificar los requerimientos de un sistema [2].

.

El uso de Método Formal propicia la confiabilidad y la seguridad de un sistema, al aumentar la comprensión acerca de un sistema revelando inconsistencias, ambigüedades e in-completitudes, que de otra manera pasan desapercibidas [1].

Los métodos formales que se utilizan para desarrollar sistemas de computadoras son técnicas de base matemática para describir las propiedades del sistema. Estos métodos formales proporcionan marcos de referencia en el seno de los cuales las personas pueden especificar, desarrollar y verificar los sistemas de manera sistemática, en lugar de hacerlo ad hoc [1].

B. VDM++

El Vienna Development Method (VDM), en español, «Método de Desarrollo de Viena» es un método formal lenguaje de especificación formal [1,2]

1. Definición de Clases

De acuerdo a [6] los modelos en VDM++ consisten en un conjunto de clases. Una clase representa una colección de objetos que comparten elementos comunes como atributos u operaciones. La estructura de la descripción de una clase se muestra en la Fig. 1. La clase se representa con la palabra reservada class, seguida por el nombre de la clase. La descripción consta de varios bloques, precedidos por la palabra reservada que indica el tipo de elemento descrito en dicho bloque. En figura 1 podemos apreciar que una clase en VDM++ tiene los siguientes bloques: a) Variables de Instancia (instance variables), las cuales modelan el estado interno del objeto. b) Tipos (types): presenta la definición de tipos de datos. c) Valores (values): permite la definición de constantes. d) Operaciones (operations): define operaciones que pueden modificar las variables de instancia.[3]

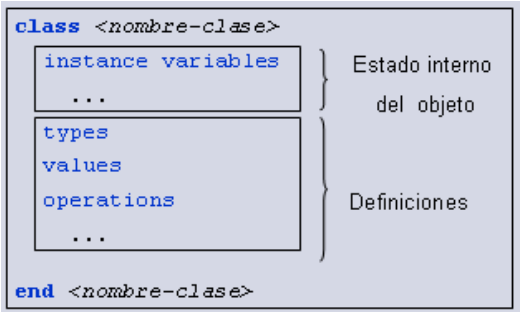

Fig. 1. Estructura de clase en VDM++

2. Tipos

En cuanto a los tipos de dato, VDM++ cuenta con tipos básicos y tipos compuestos. Entre los tipos básicos se tiene: boleanos (bool), naturales (nat, nat1), reales (real) y caracter (char). En los tipos compuestos presenta: conjuntos (set of), secuencias (seq of), mapeos (map to), entre otros. Cada uno de estos tipos tiene operaciones pre-definidas. Para la aplicación de nuestro trabajo queremos resaltar las operaciones pre-definidas para el tipo secuencia mostradas en la figura 2 [3].

Operador	Semántica	Tipo
hd l	primer elemento de la lista	seq1 of A -> A
tl l	cola de la lista	seq1 of A -> seq of A
len l	largo de lista	seq of A -> nat
inds l	índices de la lista	seq of A -> set of A
l**(i)**	elemento en la i-ésima posición	seq of A * natl -> A
l1 = l2	compara si dos listas son iguales	seq of A* seq of A ->bool

Figura. 2. Operaciones pre definidas sobre el tipo secuencia

3. Expresiones

Las expresiones son utilizadas para describir cálculos que no producen efectos secundarios, esto significa que nunca podrán afectar el valor de una variable de instancia (a menos que contenga un llamado a operación). Las expresiones pueden ser evaluadas mediante el remplazo del identificador utilizado en la expresión con valores actuales. La evaluación de una expresión da como resultado un solo valor. VDM++ tiene 25 categorías diferentes de expresiones [6]. Una de las categorías más utilizadas para la definición de pre-condiciones, post-condiciones en invariantes son las expresiones cuantificadoras.[3]

4. Expresiones cuantificadoras

Las expresiones cuantificadoras son un tipo de expresión lógica. Son utilizadas de forma frecuente cuando es necesario realizar una aserción acerca de una colección de valores. Existen dos tipos de expresiones cuantificadoras: cuantificador universal (forall) y cuantificador existencial (exists). Ambas enlazan una o más variables a un tipo de VDM++ o, a un valor que pertenece a un conjunto, y los evalúa contra una expresión booleana [3].

5. Invariantes

Si las variables de instancia especificadas en una clase contienen valores que no deberían ser permitidos, entonces es posible restringir dichos valores por medio de invariantes. El resultado será que el tipo es restringido a un subconjunto de los valores originales [6,8]. Para especificar una invariante se utiliza la palabra reservada inv a continuación de la definición de todas las variables de instancia declaradas en la clase.[3]

6. Operaciones

En VDM++ los algoritmos son definidos por operaciones. Las operaciones pueden manipular tanto variables globales y variables locales. Las operaciones pueden ser definidas de forma explícita (mediante un algoritmo explícito) o de forma implícita (mediante el uso de pre-condiciones y post-condiciones). Para poder ejecutar nuestras operaciones en el intérprete de la herramienta, las operaciones deben ser definidas explícitamente. VDM++ también permite agregar pre-condiciones y post-condiciones en las operaciones explícitas [6]. La idea principal de las pre-condiciones y de las post-condiciones es que una clase y sus usuarios tienen un "contrato". El usuario debe garantizar ciertas condiciones antes del llamado a la operación (pre-condición) y por su parte la clase garantiza que ciertas propiedades serán verdaderas cuando termine la operación (post-condición) [3].

III. VDMTOOLBOX

A. *Definición*

Es una herramienta integrada que soporta el análisis de modelos VDM++ vía análisis sintáctico, análisis de tipos, generación de condiciones de integridad y análisis de cobertura, además que permite la generación de código Java y C, en la figura 3 mostramos la ventana inicial con las opciones principales de la herramienta descritas anteriormente. La herramienta además posee un intérprete que permite validar las especificaciones. La herramienta también permite convertir directamente diagramas de clase UML creados en Rational Rose en especiación VDM++.[3]

Una vez realizada la especificación en VDM++, es necesario verificar si las clases concuerdan con las reglas sintácticas de VDM++. El analizador sintáctico verifica si la sintaxis de la especificación es correcta. El analizador sintáctico reporta los errores, si los hubiera, y realiza supuestos sobre cual pudo haber sido el error. Luego que la especificación ha pasado el analizador sintáctico, debe verificarse la correctitud de los tipos de acuerdo a las reglas de VDM++. Habiendo garantizado tanto la correctitud sintáctica como la correctitud de tipos, VDM++ToolBox provee soporte para validación de la especificación a través de la ejecución de pruebas utilizando un intérprete. El intérprete mostrado en la figura 4 permite ejecutar partes de la especificación utilizando valores seleccionados por el desarrollador (casos de prueba)[3].

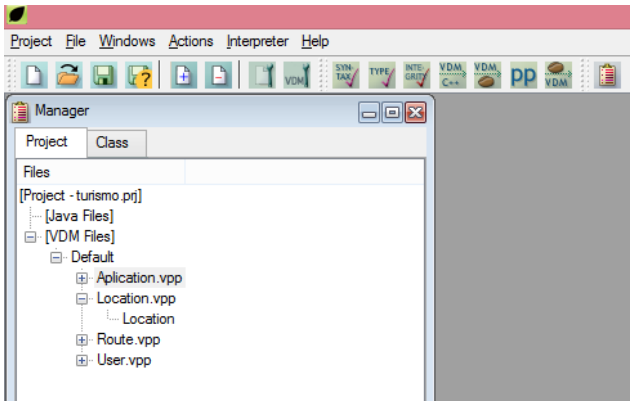

Figura 3 .Entorno de Desarrollo de VDMToolBox

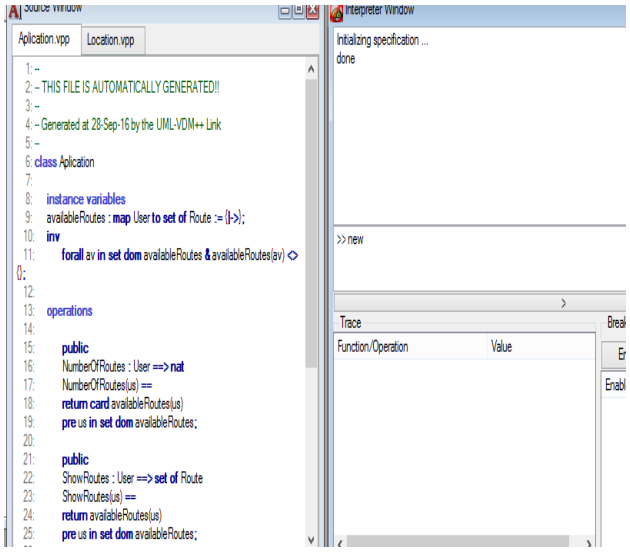

Figura 4. Vista del OutLine de DVMToolBox

B. *Casos de Uso Reales con VDMToolBox*

El desarrollo de especificación formal para sistemas con la herramienta VDMToolBox es más amplia que con Overture, esto debido al poco conocimiento de la segunda herramienta. Algunos proyectos que han aplicado VDMToolBox son:

En el trabajo de Souma y Hatayana[6] se presenta el modelamiento y validación del calculo de tarifas d tren utilizando VDM++. Se resaltana los resultados y experiencias ganadas en un Proyecto de investigación colaborativa entre AIST y OMRON, en el que VDM ++ ha sido aplicado para formalizar las especificaciones de TFCAS y validar su consistencia y propiedades de confiabilidad.[6]

El trabajo de Nicholls[7] realizó una herramienta de Validación Gráfica para un Marcpaso.. En su artículo se diseñó e implementó una interfaz gráfica de usuario para usar en la validación de un modelo formal de un marcapasos cardíaco. Los modelos se ejecutan utilizando el intérprete VDMTools y la comunicación con la GUI de validación se lleva a cabo a través de la API CORBA del Toolkit. Se requerían algunos cambios en el modelo para facilitar esta comunicación: estos cambios se llevaron a cabo con éxito. Se diseñó un editor de escenarios para los modelos pero no se implementó debido a limitaciones de tiempo [7]

El trabajo de Adelard [8] describe un enfoque para el uso del método formal VDM en el diseño y la implementación de interfaces Microsoft WindowsTM. Este enfoque se desarrolló durante el desarrollo de Dust-ExpertTM, un sistema basado en Windows. [8]

IV. OVERTURE TOOL

A. *Definición*

Overture Tool es un entorno de desarrollo integrado (IDE) mostrado en la figura 5, de código abierto para desarrollar y analizar modelos VDM. La suite de herramientas está escrita enteramente en Java y construida sobre la plataforma Eclipse[4].

La misión de Overture es doble:

• Proporcionar una herramienta de fuerza industrial que apoye el uso de modelos abstractos precisos en el desarrollo de software,

• Fomentar un entorno que permita a los investigadores y a otras partes interesadas experimentar con modificaciones y ampliaciones de la herramienta [5].

En la figura 6 mostramos la estructura genrada a partir de codigo en overture.

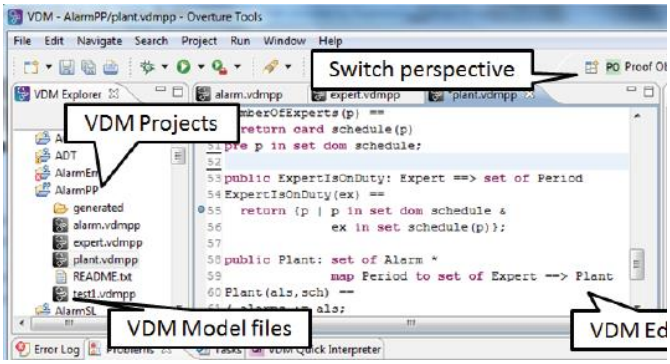

Figura 5. Entorno de Desarrollo Overture Tool

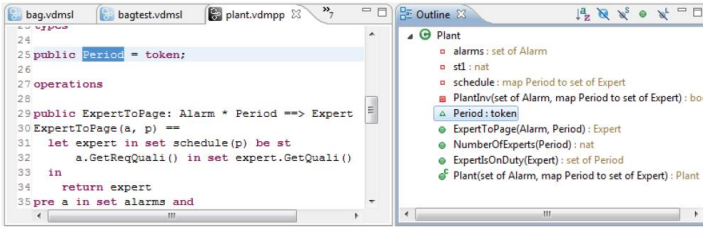

Figura 6. Vista del OutLine de Overture Tool

B. *Casos de Uso Reales con Overture*

La generación de especificaciones con overture está en una etapa evolutiva, los desarrolladores de software empiezan a interesarse en esta nueva herramienta de leguaje de especificación formal con VDM, algunos de los proyectos que empiezan a validar sus requerimientos son .

La investigación de Omori y Araki [9] muestra el uso de Overture para manejar requerimientos formales. Su trabajo propone un proceso evolutivo de un viaje de ida y vuelta entre la especificación pre-formal y la especificación formal con la herramienta

Mascil [10] presenta un entorno gráfico para modelado y prototipado de sistemas interactivos. Los prototipos desarrollados dentro de PVSio-web pueden parecerse mucho a la apariencia visual y al comportamiento de un sistema real. El comportamiento de los prototipos está totalmente impulsado por modelos formales ejecutables[10].

En Verhoef y Perrotin[11] muestran las herramientas TASTE y Overture, los autores han demostrado con éxito que la integración de métodos es una estrategia muy prometedora para crear suites de herramientas robustas y eficaces. Exploramos el camino de la integración de ambas suites de herramientas de código abierto, para aprovechar aún más su eficacia, con el objetivo de proporcionar una plataforma potencial para el diseño de extremo a extremo y la ingeniería de sistemas embebidos fiables, centrándose en la aviónica de la nave espacial.

V. CASO DE ESTUDIO

Para mostrar el uso de ambas herramientas y su posterior comparación se ha utilizado un caso de estudio que corresponde al desarrollo de un sistema de: LOCALIZACION PARA TURISTAS. EXPLICAR EL SISTEMA TAL CUAL LO EXPUSIERON EN CLASE. MOSTRAR EL MODELO DE CLASES COMPLETO Y LUEGO SE DICE QUE POR MOTIVOS DE ESPACIO SOLO SE TRABAJAR LA CLASE LOCATION

Se ha utilizado el lenguaje VDM en el dialecto VDM++ que extiende a VDM-SL brindándole características para el modelado orientado a objetos y para la concurrencia.

Se ha realizado la especificación formal de una clase Location descrita en la figura7.

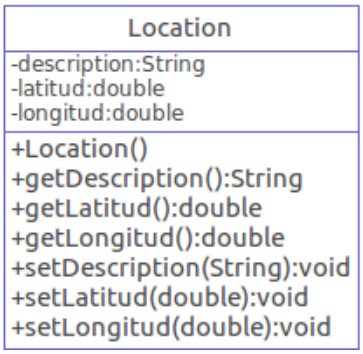

Figura 7. Diagrama de Clases de Location

Dicha clase representa la ubicación geográfica de un determinado punto. Para ello cuenta con valores para longitud y latitud, y una descripción adicional.

Dicha clase tiene los siguientes atributos representados en la figura 8:

- description: cadena de caracteres,
- longitud: real,
- latitud: real.

```
class Location
  types
    public string = seq of char;
  instance variables
    description : string := "";
    longitud : real := 0.0;
    latitud : real := 0.0;
```

Figura 8 . Class Location en lenguaje VDM

Se han definido las variantes para que las coordenadas no pierdan su integridad a lo largo de la puesta en marcha de la clase. En primer lugar se aseguró que el componente longitud de la coordenada se encuentre entre los límites -180 y 180. Del mismo modo, el componente latitud se debe encontrar en el rango de -90 a 90(Fig. 9).

```
  inv
    latitud <= 90;
  inv
    latitud >= -90;
  inv
    longitud <= 180;
  inv
    longitud >= -180;
```

Figura 9. Descripción de invariantes en Location

Adicionalmente se han definido 6 funciones: 3 getter y 3 setter para cada atributo (Fig. 10).

```
operations

  public
  GetDescription: () ==> string
  GetDescription() == return description;

  public
  GetLongitud: () ==> real
  GetLongitud() == return longitud;

  public
  GetLatitud: () ==> real
  GetLatitud() == return latitud;
```

Figura 10 . Métodos get para la clase Location

En el setter de longitud y latitud se aseguró que el nuevo valor de cada uno, se encuentre en el rango de valores válidos descritos anteriormente utilizando para ello precondiciones. En el caso del atributo description, fue descrito utilizando una precondición que garantice que la longitud del nuevo valor no excede los 50 caracteres (Fig. 11).

```
  public
  SetDescription: string ==> ()
  SetDescription(d) == description:= d
  pre len d < 50;
```

Figura 11. Precondicion para setDescription

VI. PRUEBAS

A. *Aplicando VDMToolBox*

En la figura 12 y 13 se presentan la generación de código utilizando VDMTools++.

El nivel de acceso de los atributos es private y han sido inicializadas con 0.0 (longitud y latitud) y con cadena vacía (descripción) (líneas 33, 37, 41 de la Figura 12). Se puede observar que la secuencia de caracteres de VDM se ha convertido en un String en el archivo Java y los tipos real se han sido traducidos a tipo Double (Fig. 13).

```
public class Location {

// ***** VDMTOOLS START Name=vdmComp KEEP=NO
  static UTIL.VDMCompare vdmComp = new UTIL.VDMCompare();
// ***** VDMTOOLS END Name=vdmComp

// ***** VDMTOOLS START Name=description KEEP=NO
  private String description = null;
// ***** VDMTOOLS END Name=description

// ***** VDMTOOLS START Name=longitud KEEP=NO
  private Double longitud = null;
// ***** VDMTOOLS END Name=longitud

// ***** VDMTOOLS START Name=latitud KEEP=NO
  private Double latitud = null;
// ***** VDMTOOLS END Name=latitud
```

Figura 12 . Atributos privados inicializados con null.

```
    try {

      description = new String("");
      longitud = UTIL.NumberToReal(new Double(0));
      latitud = UTIL.NumberToReal(new Double(0));
    }
```

Figura 13 . Atributos con valores de acuerdo a su tipo

El generador ha creado los setter y getter adecuadamente, además existen controles para las variantes, precondiciones, pos-condiciones definidas en el archivo vpp (Fig. 14). El generador adicionalmente ha creado un constructor que ejecuta un método para garantizar la integridad de las invariantes durante la instanciación de objetos de la clase Location. En la Figura 15 se muestra el código generado del setDescription, en el cual se asegura que que la longitud del nueo atributo cumpla con la condición de no tener un lenght mayor a 50. Los demás setter y getter son generados similarmente.

```
// ***** VDMTOOLS START Name=pre_SetDescription#1|String KEEP=NO
  public Boolean pre_SetDescription (final String d) throws CGException {
    return new Boolean((d.length()) < (50));
  }
```

Figura 14 . Control de precondición de setDescription.

```
// ***** VDMTOOLS START Name=SetDescription#1|String KEEP=NO
  public void SetDescription (final String d) throws CGException {

    if (!this.pre_SetDescription(d).booleanValue())
      UTIL.RunTime("Run-Time Error:Precondition failure in SetDescription");
    description = UTIL.ConvertToString(UTIL.clone(d));
  }
// ***** VDMTOOLS END Name=SetDescription#1|String
```

Figura 15 . setDescription verifica la precondición.

B. *Aplicando OvertureToolBox*

El nivel de acceso de los atributos es private y han sido inicializadas con 0.0 (longitud y latitud) y con cadena vacía (descripción). Se puede observar que la secuencia de caracteres de VDM se ha convertido en un String en el archivo Java y los tipos real se han sido traducidos a tipo Number (Fig. 16).

```
7  public class Location {
8    private String description = "";
9    private Number longitud = 0.0;
10   private Number latitud = 0.0;
```

Figura 16 . Declaración de variables como privadas.

El generador ha creado los setter y getter adecuadamente, sin embargo, no existe ningún control para las variantes, precondiciones, post-condiciones definidas en el archivo vpp (Fig. 17).

```
22   public Number GetLatitud() {
23
24     return latitud;
25   }
26
27   public void SetDescription(final String d) {
28
29     description = d;
30   }
```

Figura 17 . Getters y setters definidos por cada atributo.

En la Fig. 18, se observa que el generador adicionalmente ha creado un constructor por defecto y un método toString que imprime todos los atributos de la clase Location.

```
22   public Number GetLatitud() {
23
24     return latitud;
25   }
26
27   public void SetDescription(final String d) {
28
29     description = d;
30   }
```

Figura 18 . Constructor anónimo y método toString()

VII. Análisis de Resultados

De acuerdo a las pruebas realizadas, se presenta la Tabla 1 con un análisis comparativo entre las herramientas evaluadas para determinar cuál es la más óptima en el proceso de generación código a partir de especificación formal utilizando el lenguaje VDM:

TABLA 1 : ANÁLISIS COMPARITIVO

Característica	**VDMToolBox++**	**OvertureToolBox**
Completitud	Genera código con todas las precondiciones y postcondiciones.	Genera código con pocas precondiciones y postcondiciones.
Readable	Código difícil d leer. Exceso de comentarios: al inicio y fin de cada atributo y método.	Código fácil de leer. No genera comentario alguno.
Similitud	Código generado no muy similar al código original de VDM.	Código generado muy similar al código original de VDM.
Debug y compilación	Relativamente sencillo realizar ambas tareas, sin embargo necesitan librerías específicas (jp.co.csk.vdm.toolbox .VDM).	Relativamente sencillo realizar ambas tareas, sin embargo necesitan librerías específicas (org.overture.codegen.runtime).
Trazabilidad	Trazabilidad hacia el código original VDM completa pero difícil.	Trazabilidad hacia el código original VDM incompleta .

VIII. Conclusiones

Ambas herramientas, VDMTools++ y OvertureTool, dan la posibilidad de generar código Java (VDMTools++ además puede generar código C++) a partir de un archivo vpp (que almacena código vdm) que describe especificaciones formales.

Overture genera un código Java más limpio, más legible y fácil de testear en comparación con el código generado por VDM++. Sin embargo, no refleja las condiciones definidas en invariantes, precondiciones, pos-condiciones y las ignora en el código generado. A diferencia, VDMTools++ utiliza todas las aserciones y las refleja en el código, lo que le da mayor confiabilidad pues tiene dentro del código todas las invariantes, precondiciones y post-condiciones evitando violaciones de integridad e inconsistencia con las especificaciones formales diseñadas.

Para personas que contemplen la posibilidad de utilizar herramientas de especificación formal, deben saber que dos de las principales herramienta ofrecen la posibilidad de generar código Java, lo que implica ahorro de tiempo y esfuerzo. Podemos recomendar OvertureToolBox ya que genera código legible , sin abuso de comentarios que podrían entorpecer su lectura, además ofrece fácil trazabilidad hacia el código VDM original, sin olvidar que los procesos de compilación y depuración son sencillos de realizar.

AGRADECIMIENTOS

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g.” Try to avoid the stilted expression, “One of us (R. B. G.) thanks …” Instead, try “R.B.G. thanks …” Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCIAS

[1] Vieena Development Method, https://es.wikipedia.org/wiki/Vienna_Development_Method ,Marzo 16, 2013.

[2] Manuel I.Capel, Métodos Formales en la Ingeniería del Software, 2004..

[3] Elizabeth Vidal-Duarte, Aplicación y Validación de Especificaciones Formales Ligeras en el Modelo Conceptual: Reduciendo la ambigüedad e incrementando la Conformidad entre los Requerimientos y el Código, 2012.

[4] overturetool, http://overturetool.org/ , 2008B. Simpson, et al, “Title of paper goes here if known,” unpublished.

[5] overturetool, https://github.com/overturetool/overturetool.github.io/blob/master/community/index.md , 2010..

[6] Daisuke Souma, Goro hatayama, Hitoshi Ohsaki, Nguyen Van Tang, Modeling and Validating the Train Fare Calculation and Adjustment System Using VDM++.

[7] Emma Nicholls, A Graphical Validation Tool for a VDM++ Model of a Cardiac Pacemaker , August 2007.

[8]] Tim Clement Adelard, The formal development of a Windows interface , November 15, 1999

[9] Yoichi Omori, Keijiro Araki , and Peter Gorm Larsen, JODTool on the Overture Tool to manage formal requirement dictionaries , june 2015

[10] Paolo Masci1, Luis Diogo Couto , Peter Gorm Larsen , and Paul Curzon ,Integrating the PVSio-web modelling and prototyping environment with Overture, , June 2015

[11] Marcel Verhoef Maxime Perrotin ,TASTE for Overture to keep SLIM, june 2015