A Formal Model for the Requirements Diagrams of SysML

F. Valles-Barajas

***Abstract*— There are several notations to build a model: textual, graphical and mathematical. There are several notations to build a model: textual, graphical and by using mathematics. The Object Management Group (OMG) has developed a graphical notation to model systems called SysML (system modeling language); this notation includes the modeling of the system requirements. In this paper a precise model of the requirements diagrams of SysML is presented. This model is made using a modeling language called Alloy, which has been used to formally specify UML diagrams.**

***Keywords*— Formal methods, Alloy, SysML, requirement diagrams, system modeling.**

I. Introducción

UN MODELO sirve para documentar, comunicar y entender un sistema y constituye una vista simplificada del sistema que representa [19]. Existen diferentes notaciones para construir un modelo: textual, gráfica y matemática. Una de las notaciones gráficas que ha tomado mucha importancia en los últimos tiempos es SysML, la cual es de utilidad para modelar sistemas [1, 16]. Estos últimos están formados por hardware y software. SysML sustituyó a UML en la modelación de sistemas. Este último nació exclusivamente para el diseño de software. A pesar de que UML ha sido adaptado para modelar sistemas, para algunos diseñadores estas modificaciones no contemplan todo el potencial necesario para modelar un sistema completo [6].

El OMG (Object Management Group) tomó como base a UML para el desarrollo de SysML. De hecho SysML contiene los siguientes diagramas de UML: diagramas de máquinas de estado, diagramas de casos de uso y diagramas de interacción. Los diagramas de actividades y de clase de UML fueron modificados en SysML para adaptarlos a la modelación de sistemas. Por último los siguientes diagramas fueron creados por los desarrolladores de SysML: diagramas de requerimientos, diagramas paramétricos y diagramas de asignación.

Como su nombre lo indica, los diagramas de requerimientos se utilizan para modelar los requerimientos de un sistema.

En la actualidad existen herramientas comerciales y libres para modelar los requerimientos de los sistemas. Estas herramientas modelan los requerimientos utilizando tablas para guardar información relevante de los requerimientos (como por ejemplo: riesgos identificados e importancia de los requerimientos). Hay herramientas que modelan los requerimientos de manera gráfica utilizando la notación de UML. La ventaja de modelar los requerimientos utilizando un diagrama de requerimientos es que estos son representados como elementos de modelado y pueden ser ligados a otros elementos de modelado (por ejemplo un elemento que satisface a un requerimiento) mediante las asociaciones que permiten los diagramas de requerimientos de SysML [2, 17].

Motivación del artículo:

Lo notación de los diagramas de requerimientos está definida en la especificación del OMG [16]. En ese documento se especifican los conceptos de los diagramas de requerimientos utilizando diagramas de UML y lenguaje natural. Debido a que los modelos gráficos especificados en UML pueden generar ambigüedades al momento de interpretarlos (y más los modelos textuales), es necesario contar con un modelo formal de los diagramas de requerimientos de SysML. OCL (Object Constraint Language), es un lenguaje formal que ha sido utilizado para complementar los diagramas de UML, obteniendo de esta forma modelos más precisos y libres de ambigüedades [18].

En este artículo se utiliza Alloy para especificar la semántica y la sintaxis de los diagramas de requerimientos de SysML. Alloy es un lenguaje de modelado que permite especificar sistemas de una manera precisa, sin ambigüedades y más simple que en OCL [8].

Es importante mencionar que la OMG presenta una especificación formal para los diagramas de UML utilizando OCL, sin embargo tiene la desventaja de que es un lenguaje de especificación formal que genera modelos demasiado complejos [3, 22].

Trabajos relacionados:

Debido a que muchos autores consideran que la especificación textual del lenguaje de modelado UML puede ocasionar errores de interpretación, algunos investigadores de la comunidad científica se han avocado a la tarea de especificar UML de una forma precisa y que no de lugar a interpretaciones erróneas. Por ejemplo en [11] se presenta un metamodelo de los diagramas de clase de UML utilizando Object-Z, el cual es un lenguaje de modelado que utiliza lógica de predicados y la teoría de objetos [14] para modelar sistemas.

F. Valles-Barajas es profesor de tiempo completo en el Departamento de Tecnologías de la Información de la Universidad Regiomontana, Nuevo León, México. fernando.valles@acm.org, fernando.valles@ieee.org.

En el trabajo presentado en [9] los autores utilizan un lenguaje que es una extensión del utilizado para representar expresiones regulares. La ventaja de esa representación es que se puede verificar formalmente si un modelo hecho en UML es consistente.

La especificación formal de los diagramas de casos de uso y de los diagramas de estados utilizando Alloy se presentan respectivamente en [20] y [21].

En [10] se presenta un lenguaje algebraico llamado cálculo de actividades para especificar formalmente los diagramas de actividades de SysML. Mediante este lenguaje se puede verificar si los diagramas de actividades cumplen con todas las reglas especificadas en el documento oficial de la OMG [16]. Es importante mencionar que los diagramas de actividades ya existían en UML y fueron adoptados en SysML. En las versiones 1.x de UML, estos diagramas estaban muy relacionados a los diagramas de estados de los cuales existe literatura que formaliza su sintaxis y semántica (ver por ejemplo [21]). En la versión 2.0 de UML los diagramas de actividades dejaron de estar relacionados con los diagramas de estados y por lo tanto todos las definiciones formales que fueron hechas de la versión 1.x de UML ya no aplican. Lo interesente de [10] es que formaliza los diagramas de actividades en base a la versión 2.0 de UML.

El trabajo presentado en [4] contiene un metamodelo de los requerimientos de un sistema. Este metamodelo está compuesto de un modelo gráfico y uno formal. En el modelo gráfico se utilizó UML y en el modelo formal lógica de predicados. El modelo formal se utiliza para definir sin ambigüedades las relaciones entre requerimientos y otros elementos.
Es importante mencionar que el metamodelo no fue basado en SysML y por lo tanto muchas de las relaciones y conceptos de los diagramas de requerimientos no son incluidos. Sin embargo los autores presentan un mapeo de las relaciones de su metamodelo a las relaciones definidas en SysML.

Los autores de [13] presentan una forma para combinar los diagramas de SysML, que representan una manera informal de modelar un sistema, con las redes de Petri, que representan una manera formal de modelar y verificar sistemas. La propuesta de los autores consiste en primero modelar el sistema utilizando los diagramas de bloques, los cuales modelan la parte estructural de un sistema, y de requerimientos de SysML. Después de esto, si se desea modelar a detalle la funcionalidad de un requerimiento se utilizan las redes de Petri. El artículo no pretende formalizar los diagramas de SysML, sino más bien complementarlos con las redes de Petri.

En [5] un lenguaje llamando 001AXES propuesto por los autores es utilizado para definir de manera formal los diagramas de SysML. Este lenguaje fue creado por los autores para obtener sistemas más confiables. Los autores proponen que un sistema sea modelado utilizando primero SysML y después el modelo resultante sea complementado con la especificación formal utilizando 001AXES. Se pretende que por medio del modelo formal se puedan encontrar inconsistencias en los modelos basados en SysML.

Al momento de escribir este artículo, la OMG no había generado aún la especificación formal de los diagramas de requerimientos del lenguaje SysML, por esta razón no fue posible compararla con la especificación formal de este artículo.

Estructura del artículo:

En esta sección se ha dado la motivación de este artículo. En la sección II se presenta una explicación de los diagramas de requerimientos. En la sección III se explican los conceptos necesarios de Alloy para el entendimiento del modelo formal de los diagramas de requerimientos. En la sección IV se presenta el modelo formal de los diagramas de requerimientos que especifica claramente su semántica y sintaxis. Por último en la sección V se ofrecen las conclusiones de este artículo.

II. Diagramas de requerimientos de SysML

Los diagramas de requerimientos son un tipo de diagramas de SysML que permiten modelar los requerimientos de un sistema de manera gráfica. Utilizando estos diagramas se pueden relacionar los requerimientos de un sistema con los elementos que implementan a dichos requerimientos. Por ejemplo se puede establecer una liga entre un requerimiento y un elemento que satisface la implementación de este requerimiento. Esta capacidad permite conocer los elementos del diseño e implementación que son afectados por el cambio de un requerimiento.
Los diagramas de requerimientos no pretenden desplazar a otras formas que existen para representarlos sino más bien pretenden complementar a las formas ya existentes.

Los requerimientos en SysML son representados utilizando el mismo símbolo utilizado por UML para representar una clase. De hecho, en la definición de SysML un requerimiento se define como un tipo de clase pero con ciertas restricciones (en la sección IV se dará una mayor explicación de estas restricciones).
Es posible asociar un identificador y un texto a un requerimiento donde el diseñador puede escribir una descripción del requerimiento en un compartimiento especial.
Los diagramas de requerimientos permiten especificar relaciones entre requerimientos. Estas relaciones se explican a continuación:

- Agregación (*composite*): se utiliza para descomponer un requerimiento complejo en requerimientos más simples. El requerimiento compuesto estará

satisfecho cuando cada uno de sus sub requerimientos lo estén.

- Refinamiento (*refine*): esta relación es útil para tener un mayor detalle de un requerimiento. Por medio de esta relación es posible detallar un requerimiento utilizando por ejemplo un caso de uso o un diagrama de actividades. Por medio de esta relación se puede establecer también el detalle de un requerimiento textual utilizando otro requerimiento.

- Copia (*copy*): cuando se desea reutilizar un requerimiento que ha sido definido en un diagrama de requerimientos en otro diagrama de requerimientos, se utiliza la relación de copia. Esta relación es útil debido a que SysML no permite que un requerimiento esté en dos diagramas de requerimientos distintos. Esta relación crea dos roles: el maestro jugado por el requerimiento del cual se hace la copia, y el esclavo jugado por el requerimiento que es una copia del maestro.

- Derivación (*derive*): en ocasiones el análisis de un requerimiento, llamado fuente, puede desencadenar la necesidad de otros requerimientos, llamados derivados.

- La relación *trace* es una relación genérica y se utiliza para establecer que dos requerimientos se relacionan de alguna manera. Normalmente esta relación se utiliza sólo si las relaciones anteriores no aplican.

Es posible especificar también relaciones entre requerimientos y otros elementos de modelado.

- Verificación (*verify*): mediante esta relación se puede especificar los casos de prueba que verifican a un requerimiento. El resultado de una prueba se puede especificar también en un caso de prueba.

- Satisfacción (*satisfy*): el elemento que satisface a un requerimiento se puede especificar mediante esta relación. Por ejemplo una computadora con ciertas características puede satisfacer un requerimiento relacionado con un sistema de control.

Los diagramas de requerimientos utilizan un elemento llamado *rationale* que sirve para justificar el porque existe una relación entre dos requerimientos o el porque es necesario un requerimiento en particular en la especificación del sistema.

Con el objetivo de entender mejor la notación utilizada en los diagramas de requerimientos, a continuación se modelarán los requerimientos de un cajero automático utilizando la notación de SysML.
Los requerimientos de un cajero automático se clasificaron utilizando la notación propuesta en [12] en la cual se dividen los requerimientos en dos áreas: requerimientos funcionales y requerimientos no funcionales (también conocidos como de calidad). Los requerimientos funcionales tienen que ver con los servicios que se espera que el sistema provea (es decir, que es lo que el sistema debe de hacer). Los requerimientos no funcionales describen restricciones adicionales que debe de cumplir el software y tienen que ver con que tan bien el sistema realiza sus funciones (es decir, que tan bien hace el sistema lo que debe de hacer). Los requerimientos no funcionales se dividen en requerimientos que tienen que ver con:

- El grado de facilidad con que un sistema se puede utilizar (esto se le conoce como *usability* en la literatura en inglés).
- La confiabilidad asociada al sistema, la cual es definida como la habilidad de un sistema de comportarse consistentemente de acuerdo a los requerimientos del usuario cuando es operado en el ambiente para el cual fue diseñado.
  La confiabilidad puede ser definida en términos del tiempo en que el sistema está disponible y puede ser expresada en por ciento.
- El desempeño del sistema.
- Requerimientos que tienen que ver con el mantenimiento y pruebas del sistema.
- Todos los demás requerimientos no considerados en los cuatro puntos anteriores.

Esta clasificación de requerimientos es conocida como FURPS+ por sus siglas en ingles F: *funcional*, U: *usability*, R: *reliability*, P: *performance*, S: *supportability* y por último el símbolo + considera el resto de los tipos de requerimientos.

A continuación se listan los requerimientos para un cajero automático utilizando la notación FURPS+.

Los requerimientos funcionales considerados para el cajero automático son:

- RF1: El cajero deberá de procesar retiros.
- RF2: El cajero deberá de poder imprimir los estados de cuenta de un cliente.
- RF3: El cajero deberá de poder ofrecer tiempo aire para teléfonos celulares.

Facilidad de uso:

- El cajero automático deberá de poder ser usado por personas daltónicas.
- El tiempo de entrenamiento para las personas de soporte técnico de deberá de ser de no más de 8 horas.
- El sistema deberá de contar con ventas de ayuda en cada menú.
- Los mensajes de error deberán de ser mostrados al usuario al menos por 30 segundos.
- Todas las pantallas deberán de ser mostradas utilizando un tipo de letra Arial de 14 puntos sobre un fondo negro.

Confiabilidad:

- Frecuencia de falla: a lo más una falla por año.
- Al reiniciar el cajero después de un error, si hay una tarjeta, se deberá de comparar el balance de esta contra la información almacenada en el banco para asegurar que tenga saldo correcto. Lo anterior solo en caso de una transacción no terminada.

Los requerimientos de desempeño son:

- El sistema deberá de validar tarjetas en no más de 3 segundos.
- El sistema deberá de validar la clave de usuario en no más de 5 segundos.
- Si no hay respuesta por parte de la computadora del banco en 2 minutos, el sistema deberá de regresar la tarjeta.
- El sistema deberá de poder manejar hasta 100 terminales.
- El sistema deberá de poder soportar hasta 30 usuarios conectados simultáneamente.

Mantenimiento y pruebas:

- Para que el mantenimiento sea fácil, todo el código del cajero automático deberá de ser escrito utilizando un estándar de codificación.

Otros:

- El cajero deberá de adaptarse a varias monedas.
- El cajero deberá de poder comunicarse con diferentes computadoras en los bancos.
- El cajero deberá de aceptar varios tipos de tarjetas.
- El sistema deberá de ser hecho en C++.
- El sistema deberá de comunicarse con el banco usando encriptación de 256-bits.

Dentro de esta clasificación podemos tener también requerimientos de seguridad y disponibilidad.

Seguridad:

- El sistema deberá de proveer máxima seguridad para evitar intrusos en el sistema.
- El sistema deberá de ser seguro contra virus.

Disponibilidad:

- El sistema deberá de estar disponible las 24 horas.

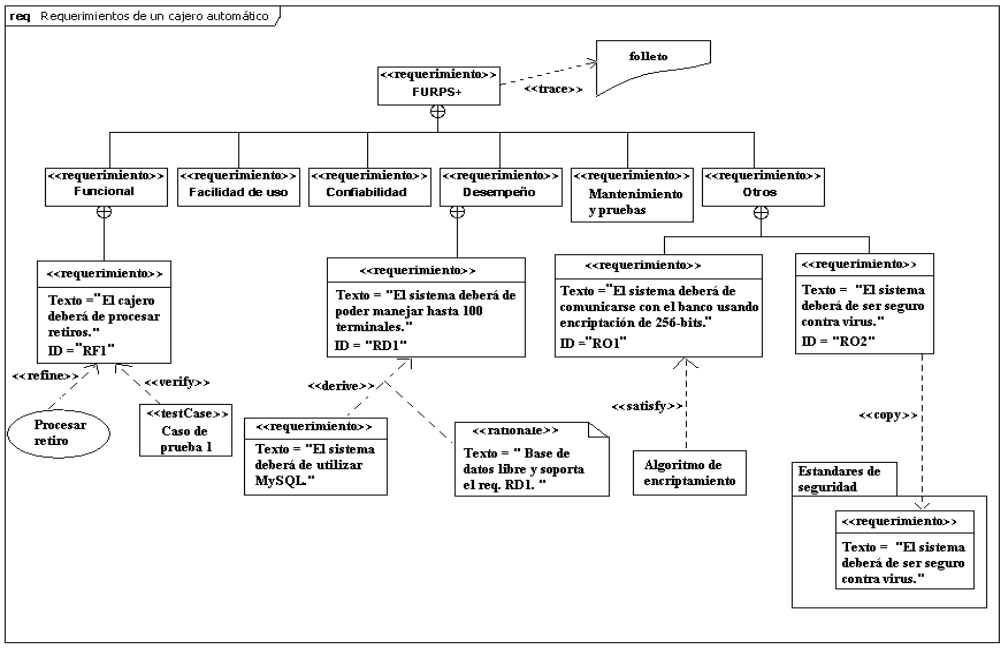

Figura 1 Diagrama de requerimientos para un cajero automático

La Fig. 1 muestra el modelo de los requerimientos del cajero automático utilizando la notación de SysML. Este modelo no está completo y solo los requerimientos que permiten la explicación de los diagramas de requerimientos fueron modelados.

Como puede verse en la Fig. 1, los diagramas de requerimientos son dibujados dentro de un rectángulo que contiene en la parte superior izquierda el nombre del diagrama de requerimientos.

En la parte superior de la Fig. 1 se definió el requerimiento FURPS+ el cual representa todos los requerimientos del cajero automático. El lector puede notar que existe una relación de *trace* entre este requerimiento y el documento "folleto" el cual representa el folleto del cajero automático que se le entrega al cliente. La relación *trace* en este caso sirve para ligar el documento que motivo la creación del diagrama de requerimientos (en este caso el folleto) con este mismo.

El requerimiento FURPS+ es un requerimiento compuesto y está dividido en requerimientos funcionales, grado de facilidad de uso del sistema, confiabilidad, desempeño, de mantenimiento y pruebas y todos los demás. Para modelar la relación entre un requerimiento compuesto y sub requerimientos se dibuja un círculo con una cruz cerca del requerimiento compuesto.

De acuerdo con la especificación del cajero automático cada sub requerimiento del requerimiento FURPS+ cuenta a su vez con varios sub requerimientos. En la Fig. 1 solo se incluyeron aquellos sub requerimientos que permitan explicar las relaciones entre requerimientos y entre requerimiento y otros elementos.

El requerimiento etiquetado como "Funcional" realmente está compuesto de tres sub requerimientos pero en la Fig. 1 solo se incluyó uno de estos. Este requerimiento sirve para ilustrar las relaciones *refine* y *verify*. Se definió el caso de uso "Procesar retiro" el cual sirve para detallar los pasos del requerimiento RF1. El caso de prueba 1 sirve para verificar si el requerimiento RF1 cumple con su especificación.

El requerimiento RD1 ilustra la relación de *derive* y el elemento *rationale*. El hecho de utilizar MySQL como base de datos se deriva del requerimiento RD1. El elemento *rationale* asignado a esta relación justifica el porque se seleccionó MySQL como base de datos.

El requerimiento etiquetado como "Otros" corresponde al símbolo + en el acrónimo FURPS+. En la Fig. 1 se definieron dos sub requerimientos de este requerimiento; RO1 y RO2. El requerimiento RO1 sirve para ilustrar la relación *satisfy* entre requerimientos y elementos de diseño o implementación. En este caso el "Algoritmo de encriptamiento" es un elemento de implementación que satisface el requerimiento RO1. El requerimiento RO2 ilustra la relación de *copy* entre requerimientos. Debido a que no es posible repetir requerimientos en los diagramas de SysML se copia el requerimiento relacionado con la seguridad contra los virus de una librería donde se definen los estándares de seguridad a la especificación del cajero automático. El requerimiento relacionado con la seguridad contra virus se definió en la librería de estándares de seguridad debido a que se espera que se utilice en más de un proyecto.

En esta sección se han explicado los diagramas de requerimientos. En la siguiente sección se explica el lenguaje de modelado utilizado para hacer el metamodelo de los diagramas de requerimientos.

III. ALLOY: UN LENGUAGE PARA CONSTRUIR MODELOS PRECISOS Y CLAROS

Alloy es un lenguaje de modelado que combina la lógica de predicados con la relacional, resultado la especificación expresada en este lenguaje precisa y sin ambigüedades [7]. Este lenguaje fue desarrollado en el M.I.T. (Instituto Tecnológico de Massachusetts) por el grupo de diseño de software. Está disponible una herramienta que permite analizar los modelos hechos en Alloy. Esta herramienta puede detectar errores de sintaxis en los modelos, pero más importante puede detectar inconsistencias en el diseño de estos modelos. Es posible generar instancias de los modelos hechos en Alloy sin tener necesidad de implementarlos en un lenguaje de programación (por ejemplo C++).

Alloy tiene sus raíces en Z (un lenguaje de modelado desarrollado en la universidad de Oxford). No es un lenguaje con tanto poder de expresión como Z pero tiene la ventaja de que se pueden generar análisis a partir de los modelos hechos en Alloy. De hecho Alloy es clasificado como un método formal ligero los cuales tienen las dos características mencionadas anteriormente.

Un sistema en Alloy es modelado especificando los estados que este puede tener así como las operaciones que hacen que un sistema cambie de un estado a otro. Alloy es un lenguaje declarativo en el sentido de que sólo se especifican las condiciones que se tienen que cumplir para pasar de un estado a otro y no se especifican los pasos que se tienen que seguir para poder efectuar dicha transición.

Alloy modela las entidades de un sistema utilizando unos elementos de modelado llamados firmas (*signatures*), los cuales son parecidos a las clases en la teoría de objetos. Se pueden generar instancias a partir de estas firmas de la misma manera que se pueden generar instancias a partir de una clase.

En la teoría de objetos cuando se modela un sistema se identifican las entidades que existen en ese sistema así como también las características de cada entidad [14]. Estas entidades son modeladas utilizando clases y sus características se especifican dentro de las clases utilizando variables. De esta misma forma, en Alloy las entidades de un sistema se modelan con las firmas y las características de cada entidad se especifican dentro de las firmas utilizando atributos.

Las relaciones entre las entidades se especifican dentro de las firmas en forma de atributos. Por ejemplo el siguiente modelo define dos entidades relacionadas:

sig A{r: B} **sig** B{}

El atributo *r* de la firma *A* define una relación de *A* a *B* de la forma *r: A →B*
Es posible restringir el número de instancias de una firma que participa en una relación mediante las siguientes palabras reservadas de multiplicidad: **some** (uno o más), **set** (cualquier número), **lone** (a lo más uno), **one** (exactamente una, este es el valor escogido por defecto). Por ejemplo si sea desea especificar que por cada instancia de *A* habrá a lo más una instancia de *B*, la relación *r* debe definirse de la siguiente forma:

sig A{r: **lone** B} **sig** B{}

Restricciones más complejas pueden definirse utilizando lógica de predicados y relacional.
Una de las ventajas de Alloy sobre otros lenguajes de modelado es que para especificar las restricciones con lógica de predicados o relacional no es necesario tener un procesador de texto especial, todos los símbolos de lógica se especifican en texto, por ejemplo los cuantificadores existencial ∃ y universal ∀ se especifican utilizando respectivamente las palabras reservadas **some** y **all**.
Alloy utiliza operadores relacionales como el operador punto "**.**" y el operador de producto cartesiano "→". Supongamos que *ia* es una instancia de *A*, entonces *ia.r* representa los elementos de *B* relacionados con *ia*. *A→B* forma una relación que contiene tuplas con elementos *(a,b)* donde *a∈A* y *b∈B*. Esta relación también puede verse como la aplicación del producto cartesiano al conjunto *A* y *B*. Cuando *A* y *B* son escalares *A→B* resulta en la tupla
(A, B).
En esta sección los conceptos principales relacionados con Alloy han sido presentados. Para una información más completa de este lenguaje de modelado véase [7].

IV. Modelo formal para los diagramas de requerimientos

En esta sección se presenta el metamodelo de los diagramas de requerimientos de SysML utilizando Alloy el cual especifica la semántica y sintaxis de los diagrama de requerimientos. Un metamodelo define las reglas de creación de modelos que se encuentran a un nivel de abstracción inferior a este y se han usado para satisfacer diferentes objetivos especificados por los desarrolladores de software. Un ejemplo de metamodelo es el desarrollado para gestionar la medición del software [15].
Para una mejor explicación se presentará de manera incremental este modelo. Se partirá de un modelo inicial que contendrá conceptos básicos de Alloy y de los diagramas de requerimientos y posteriormente se le irán agregando nuevas características de estos dos.

Las convenciones utilizadas en el modelo formal son:

- En la explicación, las referencias a los elementos del modelo se hacen en letra *itálica* para poder distinguirlos mejor.

```
1.  module sysml_models/requirements_diagram

2.  open util/boolean as B

3.  abstract sig Classifier{ isAbstract: Bool }

4.  sig Operation, Attribute{ }

5.  sig Class extends Classifier{
6.        nestedClassifiers: set Classifier,  -- relación de agregación
7.        ownedOperations: set Operation,
8.        ownedAttributes: set Attribute,
9.        superclasses, subclasses: set Class
10.       }

11.  sig String{}
12.  enum RequirementType{Funcionality, Usability,  Reliability,
     Performance, Supportability}
13.  enum RequirementRisk{High, Medium, Low}
14.  sig VerficationStatus{ }

15.  sig Requirement extends Class{
16.       id, text: String,
17.       type: RequirementType,
18.       risk: RequirementRisk,
19.       status: VerficationStatus }
20.       {
21.       isAbstract = True
22.       no ownedOperations
23.       no ownedAttributes
24.       no superclasses
25.       no subclasses
26.       }
27.  assert consistent{
28.       all r: Requirement | r.nestedClassifiers in Requirement
29.       }
30.  check consistent for 4
```

Figura 2 Modelo inicial para el diagrama de requerimientos.

- Las palabras reservadas del lenguaje de modelado Alloy son escritas en letra **negrita**.

- Los conceptos de SysML fueron escritos en inglés para que el lector pueda relacionarlos con los definidos en la especificación de la OMG.

A. Modelo inicial

La Fig. 2 contiene el modelo inicial de los diagramas de requerimientos. La línea 1 de dicha figura declara el nombre del modelo (*requirements_diagram*) y el directorio
donde este es guardado (*sysml_models*). La línea 2 importa definiciones relacionadas con variables booleanas predefinidas en Alloy. Esta línea es necesaria, por ejemplo para definir variables como la variable *isAbstract* definida en la línea 3. En

esta última línea se define una firma llamada *Classifier* que representa a un elemento de modelado de SysML. Como fue mencionado previamente una firma es parecida a una clase en la teoría de objetos [14] en el sentido de que se pueden generar instancias a partir de una firma de manera la misma forma en que se pueden generar objetos a partir de una clase.
En Alloy es posible especificar que no se generen a partir de una firma instancias agregando el modificador **abstract** al principio de una firma (este es el caso de la firma *Classifier*). Análogamente, en la teoría de objetos es posible restringir que una clase no tenga instancias.

La línea 5 especifica que la firma *Class* extiende a la firma *Classifier*. De las líneas 6 a la 9 se pueden ver las declaraciones de las relaciones entre la firma *Class* y otros elementos de modelado. Por ejemplo en la líneas 7 y 8 se declara las relaciones entre la firma *Class* y las firmas *Operation* y *Atrribute*. Se puede ver que estas declaraciones han sida hechas como campos de la firma *Class*. La palabra reservada **set** especifica que una clase puede tener cero o más operaciones y atributos.

De acuerdo a la definición de la OMG, un requerimiento especifica la capacidad o condición que un sistema debe cumplir. Un requerimiento debe de tener un identificador así como en texto donde se escriba su descripción. A un requerimiento se le puede asociar el riesgo de este (bajo, medio o alto) así como también el tipo de requerimiento (funcional, facilidad para utilizar el sistema, confiabilidad, grado de desempeño y de pruebas y mantenimiento). Es importante mencionar que los requerimientos de pruebas y mantenimiento son conocidos con *Supportability* en la literatura en inglés.
Por último, un requerimiento debe de contar con un estado.

La definición de un requerimiento se presenta en la Fig. 2 en las líneas 15-26. De acuerdo a lo anterior se puede afirmar que Alloy se pude utilizar para especificar la definición de los elementos que componen los diagramas de requerimientos. En otras palabras se puede decir que Alloy sirve para especificar la semántica de los elementos que componen los diagramas de requerimientos.

Un requerimiento en SysML se modela como una clase que está sujeta a ciertas restricciones (nótese que la firma *Requirement* ha sido definida como un tipo de *Class*). Estas restricciones se enumeran a continuación:

- No se pueden generar instancias a partir de un requerimiento. Para poder modelar esto en el modelo, la variable *isAbstract* de la firma *Classifier* se le tiene que asignar el valor de *True* (ver la línea 21). Esta variable se hereda de la firma *Classifier* a la firma *Requirement.*

- Un requerimiento no puede tener operaciones ni tampoco atributos. Las operaciones y los atributos son definidos en la firma *Class* y heredados a la firma *Requirement*. Para restringir que un requerimiento no tenga operaciones ni atributos, dentro de la firma *Requirement* se definen dos enunciados utilizado el operador relacional **no** (líneas 22 y 23), el cual restringe que el número de elementos de un conjunto sea cero.

- Un requerimiento no puede tener ancestros ni descendientes (líneas 24 y 25).

El lector puede notar que las líneas 24 y 25 restringen la manera en los elementos de un diagrama de requerimientos se puede asociar (en este caso requerimientos con requerimientos). En otras palabras Alloy permite especificar de manera clara la sintaxis de los diagramas de requerimientos.
Las líneas 27-29 definen una aserción, la cual sirve para comprobar si un modelo cumple o no con una propiedad. Esta aserción trata de comprobar que en el modelo inicial todos los requerimientos compuestos están constituidos por otros requerimientos (ver la relación de agregación en la sección II). Utilizando el comando **check** el analizador de Alloy intenta encontrar un contra ejemplo, el cual es mostrado en la Fig. 3.

Se puede ver en esta figura que hay un requerimiento etiquetado con el texto *$consistency_r*. Este requerimiento es un contra ejemplo de la aserción *consistent*. El lector puede notar que este requerimiento tiene dos sub requerimientos (*Class0*, *Class1*, *Class2*) pero estos no son requerimientos, son clases. Otro error de diseño que se puede notar en esta figura es que la clase *Clase1* tiene como sub requerimiento a ella misma, lo cual no tiene sentido.

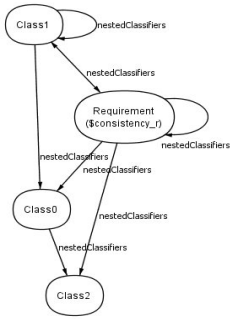

Figura 3 Instancia generada con el modelo inicial.

Para eliminar estas inconsistencias en el diseño es necesario agregar ciertas restricciones al modelo inicial. Estas restricciones se muestran en la Fig. 4.

```
1.  pred irreflexive[r: univ -> univ, A: set univ] {
2.    all a:A | a->a not in r
3.    }

4.  pred antisymmetric [r: univ -> univ, A: set univ]{
5.    all x, y: A | (x->y in r) and  (y->x in r) => x=y
6.    }

7.  pred acyclic[r: univ->univ, s: set univ] {
8.    all x: s | x !in x.^r
9.    }

10. fact compositionProperties{

11.    irreflexive[nestedClassifiers,  Classifier]
12.    antisymmetric[nestedClassifiers, Classifier]
13.    acyclic[nestedClassifiers, Classifier]
14.    }

15. fact closure{

16.    Requirement.nestedClassifiers in Requirement
17.    all c: Class, e: c.nestedClassifiers | e not in Requirement
18.    }
```

Figura 4 Restricciones agregadas al modelo inicial.

Observe que Alloy permite definir restricciones al lado de una firma (nótense que en la Fig. 2 en las líneas 6-9 se definieron restricciones relacionadas con la firma *Class*) y también restricciones fuera de una firma, como en el caso de las restricciones definidas en la Fig. 4 (líneas 10-14).

A continuación se explica a detalle la Fig. 4.

La línea 11 restingue que la relación *nestedClassifiers* sea irreflexiva. Una relación *R* sobre un conjunto *A* es llamada irreflexiva si y solo si $(a,a)\notin R$ para todo $a\in A$. Es decir relación *R* sobre un conjunto *A* es llamada irreflexiva si y solo $\forall a((a,a)\notin R)$. Debido a que *nestedClassifiers* es un campo de la firma *Class* que contiene los clasificadores que están dentro de una clase, al restringir que *nestedClassifiers* sea irreflexiva se está restringiendo que ningún clasificador se contenga a si mismo.

La línea 12 restingue que la relación *nestedClassifiers* sea antisimétrica. Una relación *R* sobre un conjunto *A* es antisimétrica si $\forall x,y((x,y)\in R\wedge(y,x)\in R\rightarrow a=b)$. Es decir, una relación *R* sobre un conjunto *A* tal que $(b,a)\in R$ y $(a,b)\in R$ solamente si $a=b$ para todo $a,b\in R$ es llamada antisimétrica.

La línea 12 restingue entonces que si una clase *c* contienen a un clasificador, este clasificador no puede contener a la clase *c*.

Para entender la restricción de la línea 13 se analizará el predicado *acyclic* (líneas 7-9) para lo cual es necesario entender los conceptos de cerradura transitiva y el operador de unión de relaciones.

La cerradura transitiva R^* de una relación binaria *R* está definida como $R^*=\bigcup_{n=1}^{\infty}R^n$. Es decir R^* está formada por los pares (x,y) tales que hay una trayectoria de longitud uno o más de *x* a *y* en *R*. R^* ofrece un mecanismo para poder saber si hay un camino de cualquier longitud entre dos puntos *x* y *y*. R^* se define en Alloy anteponiendo el símbolo de potencia a una relación (vea en el predicado *acyclic* ^*r*).

El operador punto en Alloy es el operador de unión del cálculo de relaciones. Cuando *x* es un escalar y *r* una relación, este operador regresa el conjunto de átomos que mapea *x*.

Supongamos que existe una trayectoria *t* de longitud mayor que uno que inicia en *x* y termina es ese mismo punto; es decir *t* forma un ciclo. Al calcular R^* en esta relación uno de sus tuplas tendrá la forma $(x,y,z,\ldots,x)$. Al aplicar el operador de unión entre un elemento *x* de un conjunto *s* y la relación resultante de R^* (*x.*^*r*), se generará una relación que contendrá la tupla t_1 $(y,z,\ldots,x)$. La expresión *x !in x.*^*r* del predicado acyclic restringe que *x* no este en la tupla t_1. Por lo tanto el predicado *acyclic* restringe que no haya tuplas del tipo $(x,y,z,\ldots,x)$.

Después de que las restricciones de la Fig. 4 fueron agregadas al modelo inicial no se obtuvo otro contra ejemplo para la aserción *consistent*, resolviendo con esto los problemas de diseño del modelo inicial.

El proceso que se acaba de explicar se utiliza para generar los modelos en Alloy; se crea un modelo inicial, después se revisan las propiedades de este modelo utilizando aserciones, si se encuentran inconsistencias en el diseño se modifica el modelo inicial. Para el resto del modelo se hicieron los pasos anteriores pero estos se omiten en el artículo.

B. Relaciones entre requerimientos.

La Fig. 5 muestra la especificación formal para la relación *copy* entre requerimientos. Esta figura también muestra la definición parcial de un diagrama de requerimientos. Todos los diagramas de requerimientos deben de tener al menos un requerimiento (línea 3). El lector puede notar que en las líneas 6 y 8 comienzan con -- , lo cual es una de las formas que utiliza Alloy para escribir comentarios.

```
1.  sig RequirementDiagram{
2.     heading, description: String,
3.     requirements: some Requirement,

4.     copy: set Copy }
5.     {
6.      --no pueden haber dos requirementos con un mismo id
7.      all r1, r2: requirements | r1.id = r2.id => r1 = r2
8.      --master y slave deben de estar en diagramas diferentes
9.      all c: Copy | some c.master & requirements =>
           no c.slave & requirements
10.     }

11.     fact  disjointness{
12.        -- el mismo requerimiento no puede estar en dos
13.        -- diagramas distintos
14.        all r: Requirement |
             lone {rd: RequirementDiagram | r in rd.requirements}
15.     }
16.
17.     sig Copy{
18.        master, slave: Requirement }
19.     {
20.     slave.text = master.text
21.     let nc = nestedClassifiers |
           some master.nc => some sub_s: slave.nc, sub_m: master.nc |
              sub_s.text = sub_m.text
22.     no master & slave -- &: operador de intersección
23.     }
```

Figura 5 Modelo de la relación *copy*

La línea 21 define una restricción interesante para la relación *copy*. En esta línea la palabra reservada **let** se utiliza para definir un pseudónimo para una variable. En este caso se define como pseudónimo *nc* para la variable *nestedClassifiers*. Esto permite hacer un enunciado más sencillo de entender. El enunciado de la línea 21 especifica que si en una relación *copy* el requerimiento maestro (*master*) tiene sub requerimientos, el requerimiento esclavo (*slave*) debe de tener una copia de los sub requerimientos del requerimiento maestro.

Por último un mismo requerimiento no puede ser esclavo y maestro en una relación *copy* (ver línea 22).

La Fig. 6 muestra el modelo de la relación *verify*. En esta figura se puede ver que un requerimiento puede ser realizado por una operación o por un elemento de modelado de comportamiento (línea 8). En esta definición se utiliza el operador +, que es el operador de unión de conjuntos.

```
1.  enum BehavioralModel{Interaction, StateMachine, Activity}
2.  enum VerifiedMethod{Inspection, Analysis, Demostration, Test}
3.  enum VerdictKind{Pass, Fail, Inclonclusive, Errror}

4.  sig TestCase{
5.     verifiedMethod: lone VerifiedMethod,
6.     verdict: lone VerdictKind,

7.     verifies: some Requirement,

8.     realizedBy: (Operation + BehavioralModel)
9.     }
```

Figura 6 Modelo de la relación *verify*

```
1.  sig ModelElement{}

2.  sig Satisfy{
3.     modelElement: ModelElement,
4.     req: some Requirement
5.     }


6.  sig UseCase, ActivityDiagram, TextualReq{}

7.  sig RequirementDiagram{

8.     -- líneas definidas previamente

9.     useCases: set UseCase,
10.    activityDiagrams: set ActivityDiagram,
11.    textualReqs: set TextualReq,

12.    refines: (useCases + activityDiagrams + requirements) -> (requirements
          + textualReqs)}
13.    {
14.    all r: (useCases + activityDiagrams + requirements).refines |
          r in textualReqs => refines.r in requirements
15.    }

16. sig Derive{ r1, r2: Requirement }
```

Figura 7 Modelo para las relaciones *derive*, *satisfy* y *refines*

La Fig. 7 muestra el modelo para las relaciones *derive*, *satisfy* y *refines*. En la firma *Satisfy* se especifica que un elemento de modelado puede satisfacer a uno o más requerimientos. La línea 12 define la relación *refines*, la cual define que un caso de uso o un diagrama de actividades pueden refinar a un requerimiento y que un requerimiento puede refinar a un requerimiento textual. La restricción de la línea 14 especifica que los requerimientos textuales sólo pueden ser refinados por requerimientos y no por casos de uso o por un diagrama de actividades.

Por último en la Fig. 8 se definen que no existe la relación de asociación y de herencia entre requerimientos.

```
1.  sig AssociationEnd{ attachedClass: Class }

2.  sig Association extends Classifier{
3.     e1, e2: AssociationEnd }
4.     {
5.     no (e1 + e2).attachedClass & Requirement
6.     }

7.  sig Generalization{ super, sub: Class }
8.     {
9.     (super + sub) not in Requirement
10.    }
```

Figura 8 Relaciones de asociación y herencia.

V. CONCLUSIONES

En este artículo se ha presentado un modelo formal de los diagramas de requerimientos de SysML utilizando Alloy como lenguaje de modelado. En este modelo se especificó la semántica y sintaxis de los diagramas de requerimientos.

Para modelar los elementos de construcción de un diagrama de requerimientos se utilizaron firmas de Alloy, que son parecidas a las clases en la teoría de objetos. Las relaciones entre estos elementos de construcción fueron especificadas como campos en las firmas. Las restricciones de los elementos de construcción así como de sus relaciones fueron especificadas utilizando cálculo relacional y de predicados.

Las inconsistencias en el metamodelo fueron detectadas utilizando el analizador de Alloy. Debido a lo anterior el metamodelo que se presenta en este artículo está libre de inconsistencias de diseño.

REFERENCIAS

[1] L. Balmelli, "An overview of the systems modeling language for products and systems development", *Journal of Object Technology (JOT)*, vol. 6, no. 6, pp. 149–177, July-August 2007.

[2] M. dos Santos Soares and J. Vrancken, "Model-driven user requirements specification using SysML", *Journal of software*, vol. 3, no. 6 pp. 57–68, June 2008.

[3] J. M. Fuentes, V. Quintana, J. Llorens, G. Gnova, and R. Prieto-Daz, "Errors in the UML metamodel", *ACMSIGSOFT Software Engineering Notes*, vol. 28, no. 6, pp 3–3, November 2003.

[4] A. Goknil, I. Kurtev, and K. van den Berg. A metamodeling approach for reasoning about requirements. In *ECMDAFA 2008*, pages 310–325, Berlin, Germany, June 9-13 2008. Lecture Notes in Computer Science, Springer.

[5] M. H. Hamilton and W. R. Hackler. A formal universal systems semantics for SysML. In *International council on systems engineering-2007 (INCOSE-2007)*, San Diego, California, USA, June 24-28 2007.

[6] E. Herzog and A. Pandikow. SysML: an assessment. In *15th INCOSE International Symposium*, Rochester, New York USA, July 2005. INCOSE.

[7] D. Jackson. *Software Abstractions: Logic, Language and Analysis*. MIT Press, 2006.

[8] D. Jackson and J.Wing, "Lightweight formal methods", *IEEE Computer*, vol. 29, no. 4, pp. 22–23, April 1996.

[9] S. Jansamak and A. Surarerks. Formalization of UML statechart models using concurrent regular expressions. In *Proceedings of the 27th Australasian conference on Computer science*, pages 83–88, Dunedin, New Zealand, 2004.

[10] Y. Jarraya, M. Debbabi, and J. Bentahar. On the meaning of SysML activity diagrams. In *2009 16th Annual IEEE international Conference and Workshop on the Engineering of Computer Based Systems*, San Francisco, CA USA, April 14-16 2009. IEEE.

[11] S.-K. Kim and D. Carrington. A formal mapping between UML models and Object-Z specifications. In *ZB 2000*, pages 2–21. Springer, 2000.

[12] P. Kruchten. *The Rational Unified Process*. Addison-Wesley Professional, 3rd edition, December 2003.

[13] M. V. Linhares, R. S. de Oliveira, J.-M. Farines, and F. Vernadat. Introducing the modeling and verification process in SysML. In *ETFA'2007 - 12th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, pp. 344–351, Patras, Greece, 25-28 Sept 2007. IEEE.

[14] B.Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 2nd edition, 2000.

[15] B. Mora, F. García, F. Ruiz, M. Piattini, A. Boronat, A. Gómez, J. Á. Carsí y I. Ramos., "Software generic measurement framework based on MDA", *IEEE Latin America Transactions*, vol. 6, no. 4, pp. 363-370, Aug 2008.

[16] OMG. Systems modeling language (SysML) specification. Technical report, November 2008.

[17] I. Ozkaya. Representing requirements relationships. In *First International Workshop on Requirements Engineering Visualization (REV'06)*, Minneapolis/st Paul, Minnesota, September 2006. IEEE.

[18] L. Reynoso, M. Genero, M. Piattini y E. Manso, "Un Análisis Experimental sobre el Efecto del Acoplamiento en la Comprensibilidad y Facilidad de Modificación de Expresiones OCL", *IEEE Latin America Transactions*, vol. 4, no. 2, pp. 130-135, April 2006.

[19] A. Rodríguez, E. Fernández-Medina, M. Piattini y J. Trujillo, "Secure Business Processes defined through a UML 2.0 extension", *IEEE Latin America Transactions*, vol. 6, no. 4, pp. 339-346, Aug. 2008

[20] F. Valles-Barajas. A formal model for a requirement engineering tool. In *First Alloy workshop co-located with the 14th ACM/SIGSOFT Symposium on Foundations of Software Engineering (FSE'06)*, Portland, Oregon, USA, November 2006. ACM.

[21] F. Valles-Barajas. A formal model for a state machine modeling tool: A lightweight approach. In *3rd IEEE Systems and Software Week (SASW 2007)*, Baltimore, Maryland, March 2007. IEEE.

[22] M. Vaziri and D. Jackson. Some shortcomings of OCL, the object constraint language of UML. In *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, pages 555–562, Santa Barbara, California, USA, 2000. IEEE Computer Society.

Fernando Valles-Barajas (M'2007) nació en Torreón, Coahuila México. Obtuvo un postgrado en Ciencias Computacionales en el Centro de Graduados e Investigación del Instituto Tecnológico de la Laguna. Realizó una maestría en Ingeniería de Control y un doctorado en Inteligencia Artificial en el ITESM campus Monterrey. Trabajos laborales: Ingeniero de Software, asistente de investigación en los departamentos de Mecatrónica e Ingeniería Agrícola del ITESM campus Monterrey. Obtuvo una certificación en Personal Software Process por parte del Software Engineering Institute de la universidad de Carnegie Mellon. Actualmente es profesor de tiempo completo en el Departamento de Tecnologías de la Información del la Universidad Regiomontana. Es miembro del IEEE y de la ACM.