DESARROLLO FORMAL DE LA ESTRUCTURA PRINCIPAL DE UNA APLICACIÓN (*CORE APPLICATION*) PARA UNA RED SOCIAL UTILIZANDO JML

Juan Carlos Martínez Arias
Pontificia Universidad Javeriana
Cali – Colombia
juancmartinez@javerianacali.edu.co

Néstor Cataño Collazos
Madeira ITI
Funchal, Portugal
ncatano@uma.pt

RESUMEN

Las redes sociales han cambiado de forma radical la manera de relacionarnos con otras personas y con la sociedad en general, pero estos cambios y el rápido crecimiento de estas redes han generado grandes problemas de privacidad para sus usuarios y de seguridad de la información. Posibles alternativas para solucionar los problemas de seguridad y privacidad en sistemas como las redes sociales, es la utilización de herramientas o métodos formales para desarrollar software confiable y seguro, incluyendo por ejemplo, en el mismo proceso de desarrollo, las especificaciones establecidas para la aplicación, o modelos abstractos que reflejen el comportamiento de los sistemas a implementar, mejorando así la calidad del software. Una herramienta para el desarrollo formal de software es el Java Modelling Language (JML). JML es un lenguaje de especificación formal del comportamiento funcional de aplicaciones desarrolladas en Java y permite especificar este comportamiento funcional en los métodos y clases escritas en dicho lenguaje. Otra alternativa para el desarrollo formal es el Método B. El Método-B es una metodología rigurosa para el desarrollo de software de tal forma que se garantice su correcto funcionamiento. El Método-B permite realizar desde especificaciones, refinamientos y diseño, llegando hasta la implementación y generación de código automático. Un trabajo sobre los aspectos de seguridad y privacidad en las redes sociales es presentado por los investigadores Néstor Cataño y Camilo Rueda, quienes están trabajan en un modelo formal para redes sociales denominado "**MATELAS**", el cual está basado en el cálculo B. MATELAS considera las políticas de seguridad acerca del contenido (videos, fotos, información personal, etc.), y asuntos de privacidad relacionada con el comportamiento humano (relaciones de amistad en redes sociales). A partir del modelo presentado en MATELAS y tomando como base un proceso desarrollado para la traducción de especificaciones en B a JML, presentamos la aplicación de una metodología de desarrollo formal de software que integra el uso de abstracciones y refinamientos propios del Método B con la especificación de aplicaciones Java con JML, basadas en el enfoque de "diseño por contrato". La integración de estos dos estilos de desarrollo, busca potencializar las ventajas de cada uno de ellos. La metodología propuesta se aplicó al desarrollando de la estructura principal (*core*) de una aplicación de una red social, la cual requiere que la implementación satisfaga

propiedades de seguridad y de privacidad para garantizar que no se presenten los problemas asociados a estos tipos de aplicaciones.

1. INTRODUCCIÓN

Desde la masificación del Internet, la forma de vida de las personas ha cambiado, comenzando por el modo de acceso a la información, pasando por la forma de hacer negocios hasta la manera de relacionarse. Hoy en día la proliferación de las redes sociales ha permitido a millones de personas utilizar una forma alternativa para compartir con amigos y familiares en cualquier parte del mundo sus vivencias y estilo de vida. Algunos estudios indican incluso que, las relaciones en las redes sociales ya superaron las relaciones cara a cara [(1).]

Una de las características fundamentales de estos sistemas de redes sociales es la creación por parte de los usuarios (integrantes de la red) de perfiles de cada uno de ellos, los cuales contienen información personal y la posibilidad permanente de "publicar" contenido como fotos, videos y mensajes. Algunos sitios de redes sociales como Facebook permiten también añadir pequeñas aplicaciones para compartir con sus conocidos en la red.

Hoy en día existen millones de personas que hacen parte de estas redes, solo en Facebook existen más de 500 millones de usuarios activos, donde el promedio por usuario es de 150 amigos en la red social [(2)]. Redes sociales como Twitter que se centran en el servicio denominado "microblogging", reportan un incremento en el último año del 15% al 35% de los usuarios de Internet que utilizan esta red social [(1)]. El auge de las redes sociales y la gran cantidad de material que se publica en ellas (Facebook reporta 300 billones de piezas de contenidos cargadas en los perfiles de los usuarios cada mes) hace que el tema de la privacidad y la seguridad de la información sea un aspecto muy importante a tener en cuenta.

El problema de la privacidad y seguridad de los contenidos en los sitios de las redes social se ha hecho evidente con la utilización de esta información por parte de "mercaderes" en todo el mundo [(3)], sin contar las deficiencias que existen en las aplicaciones que han permitido que se viole la privacidad de las cuentas de los usuarios [(4)].

La disponibilidad o fácil acceso de la información en las redes sociales ha generado grandes problemas a sus usuarios, desde la discriminación social por exposición de información de índole privada y personal, pasando por expulsiones de colegios, universidades y trabajos por revelar situaciones muy comprometedoras de ellos [(5)], y llegando hasta situaciones extremas como el asesinado por parte de un esposo celoso al enterarse de información publicada en el perfil de su pareja en la red social [(6)].

Para los adolecentes y jóvenes, quienes son gran parte de los usuarios de las redes sociales, éstas son espacios privados donde ellos interactúan libremente, los cuales se han convertido en parte de su vida y una de las formas principales de interacción con sus amigos y conocidos (incluso con aquellos que

no conocen). Sin embargo, para los adultos es un lugar donde la confianza e ingenuidad de los jóvenes los tienen expuestos a muchos riesgos. Un problema creciente en las redes sociales en sus aspectos de privacidad y seguridad, son los contactos de adultos con menores de edad, quienes buscan aprovecharse de la vulnerabilidad de los niños y jóvenes. Múltiples situaciones se han presentado en todo el mundo, un ejemplo de ello son las informaciones del "Center for Missing and Exploited Children" quien anunció de la existencia de más de 2.600 incidentes de adultos utilizando Internet para aproximarse online a menores con el fin de realizar algún tipo de actividad sexual [7].

Las investigadoras Danah m. Boyd de la Escuela de Información de la Universidad de California-Berkeley y Nicole B. Ellison del Departamento de Telecomunicación, Estudios de Información y Medios de la Universidad del Estado de Michigan, plantean que para muchos jóvenes se "produce una desconexión entre el deseo de proteger la vida privada y sus comportamientos" [8] y aparece lo que se denomina la "paradoja de la privacidad", la que ocurre cuando los jóvenes (y es posible que muchas personas de todas las edades) no tienen conciencia que Internet es una red pública, o teniendo presente este aspecto generan una gran confianza en el sitio con el que están interactuando.

A pesar que hoy en día se están realizando grandes esfuerzos en dar solución a estos inconvenientes, todavía subsisten problemas, e incluso, algunos afirman que estas nuevas "configuraciones de privacidad" son manipuladas con fines comerciales por los administradores de las redes sociales [9].

Posibles alternativas para solucionar los problemas de seguridad y privacidad en sistemas como las redes sociales, es la utilización de herramientas o métodos formales para desarrollar software confiable y seguro, incluyendo por ejemplo, en el mismo proceso de desarrollo, las especificaciones establecidas para la aplicación, o modelos abstractos que reflejen el comportamiento de los sistemas a implementar, mejorando así la calidad del software. Entre las herramientas para especificaciones formales se encuentra *Java Modelling Language (JML)* [10] que permite verificar el comportamiento de aplicaciones desarrolladas en Java (clases e interfaces) y entre los métodos formales se encuentra el *Método-B* [11], que permite modelar las especificaciones en "maquinas" abstractas y posteriormente poder generar un código que cumpla con las invariantes establecidas.

Como parte de un trabajo en curso sobre los aspectos de seguridad y privacidad en las redes sociales, los investigadores de las Universidades de Madeira y Pontificia Universidad Javeriana de Cali, Néstor Cataño y Camilo Rueda, trabajan en un modelo formal para redes sociales denominado MATELAS [12], el cual está basado en el cálculo B. MATELAS considera las políticas de seguridad acerca del contenido (videos, fotos, información personal, etc.), y asuntos de privacidad relacionada con el comportamiento humano (relaciones de amistad en redes sociales).

En este documento presentamos una metodología de desarrollo formal de software que integra el uso de abstracciones y refinamientos propios del Método B con la especificación de aplicaciones Java de JML basadas en el enfoque de "diseño por contrato" [13]. La integración de estos dos estilos de desarrollo, busca potencializar las ventajas de cada uno de ellos: Las especificaciones modulares y previas a la implementación de las máquinas en B, con las robustas verificaciones que se pueden

realizar sobre código Java con especificaciones en JML. La metodología propuesta se valida desarrollando la estructura principal (*core*) de una aplicación de una red social: A partir de un modelo B existente conocido como MATELAS se construye una especificación en JML y una implementación en Java. Este es un sistema no trivial en el cual se requiere que la implementación satisfaga propiedades de seguridad y de privacidad para garantizar que no se presenten los problemas asociados a estos tipos de aplicaciones.

Un aspecto que permitió implementar la metodología de forma fluida y efectiva, fue la utilización de un método de traducción de especificaciones en B a JML [14], que contempla todos los elementos relacionados con el paradigma del "paracaidista" propio del Método B y del estilo de "diseño por contrato" de JML. Finalmente, reportamos un estudio de diferentes herramientas que validamos con las especificaciones e implementación realizada.

El artículo está organizado de la siguiente forma. La sección 2 describe el modelo en B de la red Social MATELAS, en el cual nos basamos para el desarrollo del *core* de la red social especificada con JML e implementada en Java. La sección 3 describe diferentes enfoques de desarrollo y modelado formal de aplicaciones, introduciendo especialmente las características esenciales del Método B y JML, y las fortalezas que cada una de ellas tiene en los procesos de desarrollo y verificación formal de software. La sección 5 presenta la metodología empleada para la especificación e implementación de la estructura principal de una red social, la cual se caracteriza por seguir el proceso del ciclo de vida del desarrollo de software, pero ajustado al desarrollo formal de aplicaciones, utilizando las mejores características del modelado de sistemas en B, y de la especificación y verificación utilizando JML de sistemas desarrollados en Java. La sesión 6 describe diferentes pruebas realizadas sobre la especificación e implementación de la red social, estas pruebas se realizaron con herramientas de generación automática de invariantes y generadoras de especificaciones algebraicas, entre otras. La última sección presenta las conclusiones previas obtenidas en el trabajo desarrollado y los trabajos futuros relacionados con este proyecto.

2. MATELAS: Definición de un cálculo de predicado B para una Red Social [12]

MATELAS es una especificación formal modelada con el método B para una red social, que define una abstracción para los contenidos de este tipo de redes, las políticas de privacidad y las relaciones de amistad en la red y cómo éstas afectan las políticas en lo relacionado al contenido y a otros usuarios en las redes sociales.

MATELAS hace parte de un trabajo en curso sobre los aspectos de seguridad y privacidad en las redes sociales. El modelo inicial en B se ha escrito en la herramienta ***Atelier B,*** herramienta que ha sido utilizada de manera exitosa para la construcción del software del control automático de trenes del sistema de la Línea 14 del metro de París. Los planes son llevar el modelo MATELAS a ***Event B*** (una derivación del método B) y utilizar ***Rodin*** para implementar la aplicación de la red social.

MATELAS distingue cinco aspectos independientes de las redes sociales, como son: Contenido de los usuarios y aspectos de seguridad, las relaciones de amistad, contenido de los usuarios y como éstos afectan las relaciones de amistad, plug-ins externos y la interfaz del usuario.

Actualmente se han implementado seis componentes del modelo: Una máquina abstracta, cuatro refinamientos y una máquina incluida. La Figura No. 1 muestra los aspectos más importantes de cada componente.

MÁQUINA	OBSERVACIONES
Abstracción	Contenido en la pagina, Visibilidad del contenido, dueño del contenido, privilegios de acceso
Refinamiento 1	Contenido principal, campos en la página
Refinamiento 2	Contenido Obligatorio
Refinamiento 3	Pared del usuario, contenido visible en la pared, privilegios de acceso a la pared
Amigos_sociales	Relaciones de amistad
Refinamiento 4	Relaciones entre amigos, visibilidad y privilegios

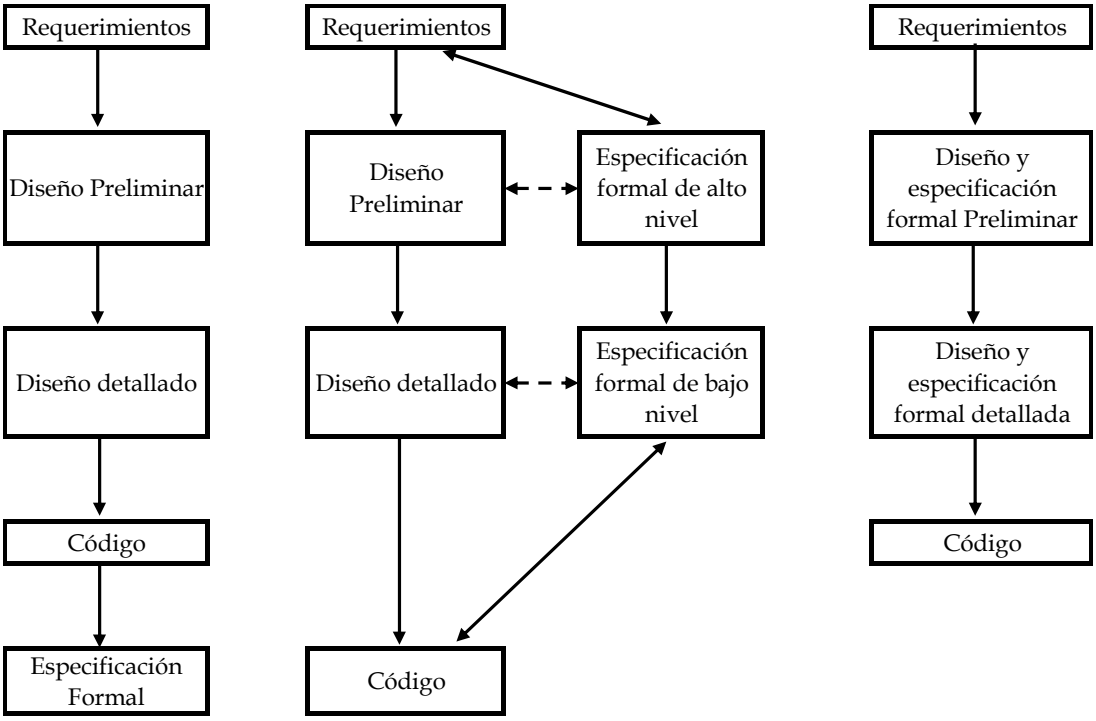
Fuente: CATAÑO, Nestor, Y RUEDA, Camilo. Matelas: A Predicate Calculus Common Formal Definition for Social Networking

Figura 1. Arquitectura del modelo MATELAS

Los creadores de MATELAS consideran dos aspectos importantes para el uso del Método B en aplicaciones de este tipo. La primera es la expresividad, que permite construir un modelo abstracto del sistema, conteniendo todas las propiedades fundamentales de seguridad y privacidad. La segunda es que las pruebas de obligación son fáciles de interpretar como predicados "antes-después" de cada operación, con lo cual es fácil descubrir posibles errores y sus correcciones.

3. ENFOQUES PARA EL DESARROLLO FORMAL DE SOFTWARE

Los métodos formales [15] son técnicas que usan nociones matemáticas para representar el comportamiento de sistemas de software y permiten especificar, diseñar y verificar el software. Con estas técnicas se pueden modelar y especificar el comportamiento de un sistema en diferentes niveles de abstracción. Sin embargo, se pueden encontrar diferentes enfoques para el desarrollo formal de software, tal como se observa en la Figura No. 2. El Método B por ejemplo, se orienta a la definición previa de todas las especificaciones del software en diferentes niveles de abstracción, su validación y una vez se garantice la correctitud de la especificación, se procede a la generación del código. JML, por el contrario, es una herramienta que permite realizar especificaciones formales sobre implementaciones Java ya desarrolladas, con el fin de validar la implementación y garantizar el comportamiento del software de acuerdo a las condiciones establecidas en las especificaciones. Cada una con fortalezas propias de su enfoque y con un alto nivel de expresividad.



Fuente: KEMMERER, Richard. Integrating formal methods into the development process. IEEE Software, 1990.

Figura 2. Comparativo de enfoques para desarrollo formal de software

En este artículo estamos presentando el desarrollo de un proyecto de software para una red social utilizando una metodología que involucra los enfoques de modelamiento y desarrollo formal propios del Método B y JML. A continuación se presente un breve resumen de estos dos diferentes enfoques.

3.1. MÉTODO B (11)

El Método-B es una metodología rigurosa para el desarrollo de software de tal forma que se garantice su correcto funcionamiento. El Método-B permite realizar desde especificaciones, refinamientos y diseño, llegando hasta la implementación y generación de código automático.

Los modelos abstractos realizados con el Método-B, son conocidos como "máquinas abstractas" o "máquinas B" y se asemejan a las nociones de "módulos" o "clases" utilizados comúnmente en programación. El método B utiliza dos conceptos esenciales [16]: el estado, que representa la noción de objetos (noción estática) y dentro de las "máquinas B" se expresan como variables y constantes del sistema, y los eventos, que representa la noción de movimiento de los objetos (noción dinámica), la cual se indicará en los modelos como las acciones sobre los estados.

Los estados dentro del modelo representan las propiedades invariantes que debe cumplir el sistema. Las invariantes son definidas en términos de las variables representadas formalmente por el cálculo de predicados y teoría de conjuntos.

Los eventos, en su noción dinámica, deberán garantizar que las invariantes se mantengan una vez ocurran. Los eventos están compuestos por "guardas", que son las condiciones necesarias para que el evento ocurra y por "acciones" que determinan cuales serán los cambios que sufran las variables y se expresan a través de operaciones.

Uno de las paradigmas usados en el Método-B para modelar un sistema es el del "paracaidista". Este paradigma consiste en realizar una observación muy general y abstracta del sistema desde lo más alto del "cielo" y a medida que el "paracaidista" va bajando, se observan hechos más específicos en el sistema. Lo anterior permite que el modelo inicial se "refine", obteniendo un modelo más concreto y finalmente, después de varios refinamientos se observaran todos los detalles del sistema. El modelo deberá garantizar que el comportamiento de la "máquina abstracta" o refinamientos anteriores se conserva en los nuevos refinamientos. Para garantizar este hecho se realizan "pruebas de obligación" que permiten validar el comportamiento de las variables y operaciones.

3.2. JAVA MODELLING LANGUAGE (JML)

El Java Modelling Language (JML) es un lenguaje de especificación formal para Java. JML utiliza el estilo de diseño por contrato (DBC por Design By Contract) de Bertrand Meyer [17], cuya idea principal es la colaboración de los elementos de un sistema de software (en Java, clases y métodos). Meyer usa la metáfora del "contrato" para representar la comunicación que debe existir entre los componentes de software a través de especificar las obligaciones y beneficios mutuos que están involucrados en estas comunicaciones. Así, se espera que se garanticen algunas condiciones (pre-condiciones) para que un componente de software ejecute una acción determinada y este componente a su vez debe garantizar que se mantengan ciertas condiciones al final de su ejecución (post-condiciones).

JML tiene una sintaxis similar a Java. Las especificaciones JML se incluyen dentro del mismo código Java utilizando los caracteres especiales para representar comentarios así: /*@ @*/ o //*@ si es en una solo línea. Este código es incluido inmediatamente antes del inicio de cada método o en el caso de las invariantes al inicio de la clase.

JML realiza las especificaciones mediante "Pre-condiciones", que indican cual debe ser el estado de un método antes de iniciarse y "Post-condiciones" que establecen cuál debe ser su estado después de la ejecución. Cuando se desea especificar el comportamiento para toda una clase se usan las "invariantes" (*invariant*), que indican una condición que debe cumplir toda la clase al inicio de ésta y al salir de ella (es posible que en la ejecución de la clase, la invariante deje de cumplirse, pero al salir debe retornar a su comportamiento deseado). Los predicados de JML son predicados de primer orden, sin "efectos colaterales" (side-effect free) [18].

Con el siguiente ejemplo de una especificación JML, se explicarán algunos conceptos e instrucciones importantes:

```
*@ public normal_behavior
        @  requires d >= 0;
        @  assignable value;
        @  ensures value == d;
        @    also
 @ public exceptional_behavior
        @  requires d < 0;
        @  assignable \nothing;
        @  signals(BankException e) e.getErrorCode() == ErrorCode.WRONG_AMOUNT;
 @*/
public Euro(double d) throws BankException {…..}
```

Figura 3. Ejemplo de una especificación con JML

En el ejemplo de la Figura No. 3 se observa como con JML se puede especificar, tanto el comportamiento normal (*normal_behavior*) que se espera del método, como algún comportamiento excepcional (*excepcional_behavior*) que pueda ocurrir. El *normal_behavior* especifica que si la pre-condición normal del método se cumple, el método debe terminar en un estado normal y se debe cumplir, por tanto, la post-condición normal. De igual forma, el *excepcional_behavior* especifica que si la pre-condición excepcional del método se cumple, el método debe terminar en un estado excepcional y se debe cumplir, por tanto, la post-condición excepcional. La especificación de la excepción se realiza utilizando la instrucción *signals*. En ambas situaciones se deben expresar las Pre-condiciones con la instrucción *requieres* y las Post-condiciones con la instrucción *ensures*.

La instrucción *assignable* (también conocida como *modifies* o *modifiable*) es utilizada para definir que variables del método podrán ser modificadas. En el caso que la instrucción *requieres* tenga el parámetro *\nothing*, significa que el método no tendrá "efectos-colaterales".

Otra característica adicional utilizada por JML son las variables tipo *model*, que permiten definir variables abstractas, las cuales son variables que existen en las especificaciones, pero no en la implementación. Si se desea relacionar las variables abstractas con variables concretas, se debe utilizar la instrucción *represents*. En JML también existen operadores como *\old*, *\result*, *\forall* y *\exist* entre otros, que tienen el siguiente uso o significado:

- *\old*, permite obtener el estado de una expresión antes de la ejecución del método en el cual es especificada. Su uso esta restringido a las post-condiciones.
- *\result*, este operador representa el valor retornado por el método. Su uso esta restringido a las post-condiciones.
- *\forall*, es la representación del cuantificador universal.
- *\exist*, es la representación del cuantificador existencial.

Finalmente, JML permite, igual que el Java, definir la visibilidad mediante los modificadores *public*, *protected* y *private*. [19]

3.2.1. Herramientas para JML

JML se ha ido convirtiendo en un "estándar de facto" [20] como lenguaje de especificación para programas realizados en Java, una de las principales razones es la gran variedad de herramientas disponibles para leer, escribir y verificar especificaciones JML.

La Universidad del Estado Iowa desarrolló un conjunto de herramientas que dan soporte al chequeo de especificaciones JML en tiempo de ejecución, *JML Common Tools*, entre las que se encuentran el jml, jmlc y jmlunit [13] [20]. El compilador JML (jmlc), fue desarrollado como una extensión del compilador de Java, y compila programas en Java especificados con JML, que incluyen en el código binario Java las instrucciones para verificar las especificaciones JML en tiempo de ejecución. La herramienta de chequeo de unidad (jmlunit), combina el compilador JML con JUnit, una herramienta de verificación para Java, que permite generar código para validar si las especificaciones JML funcionan correctamente o no.

Otras herramientas disponibles para JML [19], son ESC/java2, LOOP, JACK, Daikon, jmlspec y OpenJML. La herramienta Extended Static Cheker (ESC/java2) [21], es un validador estático de programas Java, el cual permite detectar errores comunes en el código rápidamente y verificar relativamente fácil las especificaciones JML. La herramienta LOOP, fue desarrollada en la Universidad de Nijmegen, y su propósito es convertir el código fuente Java y sus especificaciones JML, a pruebas de obligación que pueden ser validadas usando el probador de teoremas PVS (*Prototype Verification System*). La herramienta JACK, desarrollada en los laboratorios Gemplus, provee un ambiente para la verificación de programas Java y Java Card especificados formalmente con JML. La herramienta genera pruebas de obligación que pueden ser validadas usando probadores de teoremas. Daikon [22], es una herramienta que ayuda a la creación de especificaciones, infiriendo posibles invariantes con la observación del comportamiento en la ejecución de un programa. La herramienta jmlspec, puede producir una estructura de una especificación desde el código fuente de un programa Java. OpenJML [23] contiene un conjunto de herramienta para JML y permite realizar especificaciones para versiones de Java iguales o superiores a 1.5, aunque con algunas limitaciones.

4. METODOLOGÍA PARA LA ESPECIFICACIÓN Y DESARROLLO FORMAL DE UNA RED SOCIAL

Para el desarrollo formal de la estructura principal de una aplicación (*core application*) para una red social, se han integrado los enfoques para el desarrollo formal de software utilizados tanto en el método B (paradigma del "paracaidista") como en JML (Diseño por Contrato), buscando obtener los mayores beneficios de ambos enfoques, es decir, especificaciones formales previas a la

implementación que se puedan ir "refinando" a medida que se incluyen nuevos detalles en la especificación, lo anterior con el soporte de una herramienta que permita realizar las especificaciones en un lenguaje muy similar a los que los programadores "estándar" utilizan en sus desarrollos diarios en Java, implementando "contratos" entre los componentes "cliente" y "proveedores", buscando que se garantice el cumplimiento de estos "contratos".

La metodología utilizada para el desarrollo del "*core*" de la aplicación, se concibe dentro del marco de los procesos de Ingeniería de Software y específicamente en el ciclo de vida del desarrollo de software, ajustados al desarrollo formal: Obtención y análisis de requerimientos, especificación y diseño de software, implementación y validación.

4.1 OBTENCIÓN Y ANÁLISIS DE REQUERIMIENTOS

La definición y especificación de requerimientos funcionales para una red social se puede hacer con diferentes metodologías y enfoques. Para la especificación de requerimientos de la red social basaba en MATELAS, se decidió utilizar el modelo planteado por Jean-Raymond Abrial en el documento "*Mechanical Press: Requirement Document*", por ajustarse a la metodología planteada en este proyecto.

El documento está estructurado en dos bloques: La definición de los requerimientos funcionales y de seguridad, y la descripción de "operaciones" que modelan estos requerimientos funcionales. El modelo utiliza el paradigma del "paracaidista" (*Parachute Paradigm*).

En el primer bloque del documento se definen los requerimientos funcionales y de seguridad que se deben considerar en una red social estándar. Los requerimientos se etiquetan dependiendo de su tipo: FUN para los requerimientos funcionales y SEG para los requerimientos relacionados con la seguridad y privacidad de los contenidos, y su descripción se realiza en rectángulos para cada uno de los casos.

En la Figura No. 4 se observa una pequeña parte de la definición de requerimientos funcionales y de seguridad para una red social.

2. REQUERIMIENTOS FUNCIONALES

A continuación se presentan los requerimientos funcionales en una abstracción de alto nivel del sistema de una red social. Los requerimientos funcionales del sistema los llamaremos FUNCIONALIDADES y se representarán como FUN. Se establecerán los requerimientos relacionados con la seguridad y privacidad de contenidos y personas los cuales llamaremos SEGURIDAD y se representaran con SEG.

Los siguientes son los requerimientos funcionales establecidos para una red social:

FUN-1:	La red social tiene personas.
FUN-2:	En la red social existe información (contenido bruto).
FUN-3:	Las personas tendrán contenidos brutos asociados.
FUN-4:	El sistema permitirá crear personas en la red social y asociar contenidos a ellas.

Figura 4. Ejemplo de requerimientos funcionales para una red social

El modelo de Abrial también permite describir los estados (variables) y operaciones de las diferentes abstracciones y refinamientos que se realizan en un desarrollo formal basado en el paradigma del "paracaidista"'.

Se define entonces una segunda parte del documento de requerimientos que describe el modelado de los requerimientos funcionales en diferentes eventos (operaciones), desde su definición en las máquinas abstractas hasta sus refinamientos posteriores. La primera parte detalla la operación "`create_content`", la cual modela la creación de una persona en la red social con todo su contenido bruto asociado. La segunda parte describe las definiciones de la máquina abstracta "`SOCIAL_FRIENDS`" en la cual se modelan las relaciones de amistad en la red social, adicionalmente se describe la operación "`make_bestfriends`" que permite crear una relación de amistad de mejores amigos ("`best friends`") entre dos personas de la red social. Finalmente se describe la operación "`comment_rc`" en el cuarto refinamiento de la maquina abstracta "SOCIAL_NETWORK", esta operación modela la asignación de comentarios a un contenido bruto.

En la Figura No. 5 se presenta un ejemplo de la parte inicial de la descripción del primer refinamiento de la red social modelada.

3.1.5. Primer refinamiento ("SOCIAL_NETWORK_r")

3.1.5.1 Introducción

Para el primer refinamiento de la máquina abstracta se incluye una nueva variable, "principal", relacionada con la operación "create_content", cuyo fin es el de modelar agrupaciones de contenidos (*Similar a la llave primaria de una tabla en una base de datos*) asociados a un contenido bruto principal. Es este refinamiento se incluye la "acción" sobre la variable "principal" en la operación.

3.1.5.2 Refinando el estado

En este primer refinamiento del modelo se le adicionan dos (2) variables al contexto: "principal" y "field". "principal" es un subconjunto de una agrupación de contenidos brutos que la identifica. El resto de contenidos brutos de la agrupación corresponden a la variable "field" y se asocian al contenido "principal".

A continuación se presentan las nuevas variables del modelo y sus definiciones (invariantes).

VARIABLES: principal field	**inv1_1:** principal <: RAWCONTENT **inv1_2:** principal <: rawcontent **inv1_3:** principal = { } => rawcontent = { } **inv1_4:** field ∈ (rawcontent – principal) → principal

El invariante **inv1_1** indica que la variable "principal" es un subconjunto del grupo total de contenidos brutos ("RAWCONTENT") que potencialmente pueden crearse en la red social. El **inv1_2** establece que los contenidos brutos de "principal" son un subconjunto de los contenidos brutos existentes. El **inv1_3** define que no existirá contenido bruto en la red ("rawcontent"), sino existe contenido bruto definido como "principal" y finalmente el **inv1_4** establece que "field" es una relación entre un conjunto de contenidos brutos (que no son "principal") con un contenido bruto principal.

Figura 5. Ejemplo parte inicial de la descripción del primer refinamiento para una red social

La descripción de los diferentes refinamientos presentados en el documento de requerimientos describe en primer lugar las características que se adicionan a la "máquina abstracta" o refinamientos anteriores, las variables que harán parte del estado de este refinamientos, incluyendo las invariantes que deberán conservar y finalmente se describen las nuevas acciones que presentan las operaciones en cada uno de los refinamientos.

4.2. ESPECIFICACIÓN Y DISEÑO DEL SOFTWARE

La metodología utilizada para la especificación y diseño de la estructura principal de la aplicación, se realizó utilizando el enfoque del paradigma del "paracaidista" (*Parachute Paradigm*) propuesto por Abrial, utilizando JML para la especificación formal, y clases abstractas en Java para representar las máquinas abstractas y los refinamientos detallados en el documento de requerimientos.

Dado que se plantea la realización de una aplicación en Java especificada formalmente utilizando JML, que sea equivalente al modelo B existente (MATELAS), se utilizó la traducción desarrollada por los Investigadores Néstor Cataño y Tim Wahls en el documento aún no publicado "*A Baker Recipe to Port B Machines into JML Specifications*", para traducir las especificaciones B a JML.

El documento de Cataño y Wahls plantea como traducir, entre otras cosas, la definición de variables:

"*Variables in B are translated to JML* `model` *fields. For example, the variable* `person` *is translated to model* `JMLEqualsSet<Integer> person;`"

La representación de las operaciones:

"*B operations are translated to Java* `abstract` *methods.*"

Y el establecimiento de las pruebas de obligación necesarias para cada uno de los casos, por ejemplo:

"*the guard strengthening proof obligation (ANY) is FORALL(x,y).(Q(y,w) ==> P(x,v))* "

La Figura No. 6 presenta la operación "`create_content`" modela en B y la Figura No. 7 presenta la traducción a una clase abstracta en Java especificado con JML siguiendo el procedimiento de Cataño y Wahls.

```
per <-- create_content =
PRE
    person <<: PERSON & rawcontent <<: RAWCONTENT
THEN
    ANY pe , rrc
    WHERE
        pe : PERSON - person & rrc <: RAWCONTENT - rawcontent &
        rawcontent \/ rrc <: RAWCONTENT & rrc /= {}
    THEN
        person := person \/ { pe } ||
        rawcontent := rawcontent \/ rrc ||
        owner := owner \/ rrc * { pe } ||
        content := content \/ { pe } * rrc ||
        act := act \/ rrc * OPS * { pe } ||
        per := pe
    END
END;
```

Figura 6. Operación "create_content" modela en B

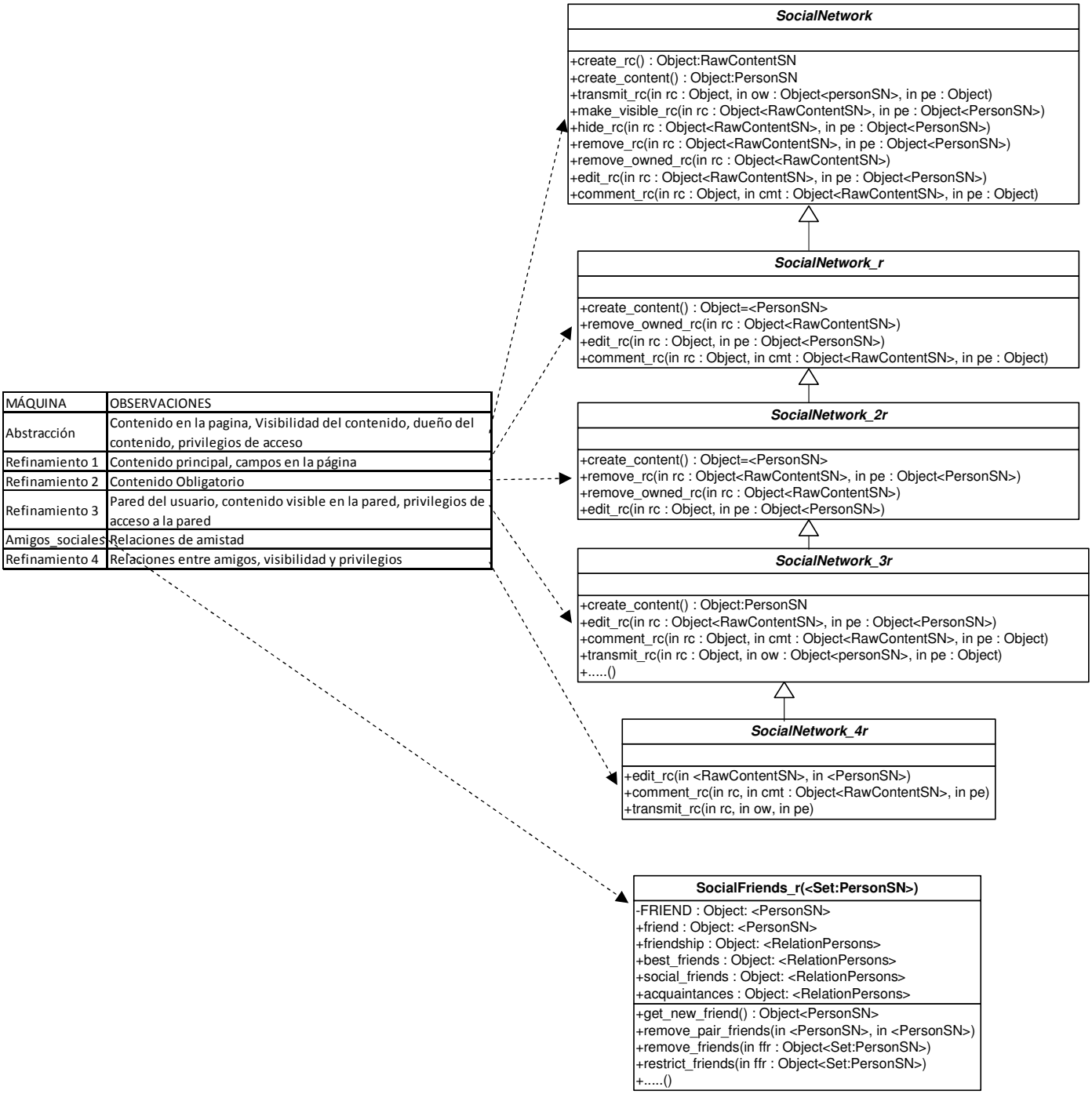
```
/*@ normal_behavior
  @      requires person_abstract.isProperSubset(PERSON_abstract) && rawcontent_abstract.isProperSubset(RAWCONTENT_abstract);
  @      assignable person_abstract, rawcontent_abstract, owner_abstract, content_abstract, act_abstract;
  @      ensures (\forall PersonSN pe; \old(PERSON_abstract.difference(person_abstract).has(pe));
                  (\forall JMLEqualsSet<RawContentSN> rrc; \old(rrc.isSubset(RAWCONTENT_abstract.difference(rawcontent_abstract))
                    && rawcontent_abstract.union(rrc).isSubset(RAWCONTENT_abstract) && !rrc.isEmpty());
                      person_abstract == \old(person_abstract.union(JMLEqualsSet.singleton(pe)))
                      && rawcontent_abstract == \old(rawcontent_abstract.union(rrc))
                      && owner_abstract == \old(owner_abstract.extendUnion(utilJML.cartesian(rrc, JMLEqualsSet.singleton(pe)).toFunction()))
                      && content_abstract == \old(content_abstract.union(utilJML.cartesian(JMLEqualsSet.singleton(pe), rrc)))
                      && act_abstract == \old(act_abstract.union(utilJML.cartesian(utilJML.cartesian(rrc,JMLEqualsSet.
                                  convertFrom(ROps.OPS.values())).toSet(), JMLEqualsSet.singleton(pe)))))
                      && \result == \old(pe)); */
public abstract PersonSN create_content();
```

Figura 7. Traducción de la operación "create_content" a una clase abstracta en Java especificado con JML

En Java, una clase que realiza un "*extends*" de una clase abstracta, se comporta como los refinamientos definidos en el Método-B, extendiendo el comportamiento especificado en una clase abstracta anterior y ampliándola o "refinándola" en la nueva clase. Esta característica del Java permitió la utilización del paradigma del "paracaidista" sin ninguna restricción en la metodología utilizada.

Teniendo como base el documento de requerimientos y el proceso de traducción de B a JML, se definieron clases abstractas en Java para realizar las especificaciones iniciales de la red social: *SocialNetwork* y *SocialFriends*, representando las "máquinas abstractas" del modelo en B, y clases abstractas que "amplian" las clases abstractas iniciales detallando las especificaciones de los métodos, representando los refinamientos definidos en la red social.

La Figura No. 8 presenta un mapeo de las máquinas abstractas y refinamientos de MATELAS, con las clases y métodos Java creadas para la especificación formal con JML.



MÁQUINA	OBSERVACIONES
Abstracción	Contenido en la pagina, Visibilidad del contenido, dueño del contenido, privilegios de acceso
Refinamiento 1	Contenido principal, campos en la página
Refinamiento 2	Contenido Obligatorio
Refinamiento 3	Pared del usuario, contenido visible en la pared, privilegios de acceso a la pared
Amigos_sociales	Relaciones de amistad
Refinamiento 4	Relaciones entre amigos, visibilidad y privilegios

Figura 8. Arquitectura de MATELAS y de la red social especificada con JML

En las especificaciones se definen variables abstractas del tipo `JMLequalsSet` [24] para representar los conjuntos definidos en el modelo en B, variables abstractas del tipo `JMLEqualsToEqualsRelation` para representar las relaciones en la red social, como las existentes entre personas y contenidos o personas y personas, y variables abstractas del tipo `JMLEqualsToEqualsMap` para representar "funciones" parciales o totales como las existentes entre

los dueños de los contenidos y éstos. Cada una de las variables abstractas definidas se asocia con sus respectivas invariantes, las cuales se deben mantener durante toda la aplicación. A continuación se presenta un ejemplo de la definición de la relación `content_abstract`, con sus respectivas invariantes:

```
// RelationContent esta definida como: person <-> rawcontent
       /*@ public model JMLEqualsToEqualsRelation<PersonSN,RawContentSN> content_abstract; */
       /*@ public invariant content_abstract.domain().isSubset(person_abstract)
                     && content_abstract.range().isSubset(rawcontent_abstract);*/
```

Las especificaciones se realizan con JML sobre clases abstractas, estableciendo la especificación como un "contrato" con precondiciones (`requires`) y postcondiciones (`ensures`), siguiendo así el estilo de diseño por contrato (*Design By Contract*) utilizado en JML.

Otro aspecto importante en esta etapa de especificación, es el establecimiento de pruebas de obligación que permitan validar que los refinamientos mantengan el comportamiento de las clases abstractas iniciales o refinamientos anteriores. Para establecer estas pruebas de obligación se siguió igualmente lo establecido en el documento "*A Baker Recipe to Port B Machines into JML Specifications*".

4.3. IMPLEMENTACIÓN

La metodología aplicada permitió definir como último refinamiento la implementación de las clases y métodos en Java, basándose en las especificaciones realizadas con JML. La implementación del "*core*" de la red social entonces, se realiza en el quinto refinamiento (*`SocialNetwork_5r.java`*). Se establece una clase (no abstracta) que "extiende" el último refinamiento, donde se definen todas las variables concretas de la aplicación y se asocian con las variables abstractas utilizadas para las especificaciones en las clases abstractas. Las siguientes instrucciones muestran como se realiza ésta asociación:

```
public HashSet<RelationContent> content;
    /*@ public represents content_abstract = contentRepresentation();

    @ public pure model JMLEqualsToEqualsRelation<PersonSN,RawContentSN> contentRepresentation(){
    @   if (this.content == null) return null;
    @   JMLEqualsToEqualsRelation<PersonSN,RawContentSN> repr = new
                      JMLEqualsToEqualsRelation<PersonSN,RawContentSN>();
    @    for (RelationContent rct : content){
    @        repr.add(rct.domainC,rct.rangeC);
    @    }
    @   return repr;
    @ }
    @*/
```

Lo que se busca es que las variables abstractas utilizadas en la especificación (ej. `content_abstract`) estén asociadas a las variables (y sus valores) definidas en la implementación (ej. `content`) con el fin que las validaciones evalúen el comportamiento de estas últimas, dentro de los métodos y clases de la aplicación.

De igual forma en esta clase (último refinamiento), se implementan todos los métodos especificados en la clase abstracta inicial y sus posteriores refinamientos, buscando que la implementación refleje todos los detalles observados en la etapa de especificación de la red social.

El segundo refinamiento de "*SocialFriends*" implementa las relaciones de amistad en la red social, las cuales serán usadas en la implementación principal de la red social.

La implementación se realizó con la versión 1.6 de Java, utilizando las nuevas características incluidas en las versione 1.5 y 1.6, en especial la definición de tipos genéricos, por ejemplo:

```
public HashSet<RelationContent> content;
```

Los tipos de datos utilizados fueron *HashSet* para representar los conjuntos de personas, contenidos y demás elementos de la red social, para representar relaciones, las cuales fueron definidas como clases en el diseño de la aplicación, y el tipo de dato *HashMap* para representar funciones totales y parciales como las existentes entre los dueños de los contenidos y éstos. La razón para utilizar este tipo de variables radica especialmente por la posibilidad de trabajar con operaciones sobre conjuntos y relaciones similares a las establecidas en MATELAS.

La siguiente es la implementación del método "remove_rc":

```
public void remove_rc(RawContentSN rc, PersonSN pe) {

        HashSet<RelationContent> tmpcontent = new HashSet<RelationContent>();
        tmpcontent.addAll(content);
        tmpcontent.remove(new RelationContent(pe,rc));
        HashSet<PersonSN> domainCt = utilSN.domain(tmpcontent);
        if (domainCt.contains(pe) && !prawcontent.contains(rc)) {
                visible.remove(new RelationContent(pe,rc));
                content.remove(new RelationContent(pe,rc));
                canvas.remove(new RelationContent(pe,rc));
                act.remove(new RelationAct(new ROps(rc,ROps.OPS.view),pe));
        }
}
```

En la implementación no se realizan especificaciones ni "contratos" pues estos quedaron definidos en la clase abstracta inicial y sus posteriores refinamientos.

4.4. VALIDACIÓN Y VERIFICACIÓN

La verificación de las especificaciones realizadas con JML y del comportamiento de la implementación de acuerdo a los "contratos" establecidos, se puede realizar, acorde a la metodología utilizada, en las etapas de especificación: validando especificaciones y pruebas de obligación, y durante la etapa de desarrollo del código Java: validando el comportamiento de los métodos con respecto a las invariantes de clase, pre y postcondiciones especificados con JML.

Para el proceso de verificación se utilizaron dos herramientas, la primera fue OpenJML [23], el cual consta de un conjunto de herramientas JML, incluyendo ESC y RAC para la verificación de especificaciones JML. Una de sus características más importantes es que permite trabajar con las versiones más recientes de Java (ej. 1.6 y 1.7), sin embargo, aún no se ha liberado la primera versión publica de la herramienta. Las verificaciones realizadas con OpenJML fueron parciales, ya que la herramienta está aún en pruebas y sus resultados no fueron los mejores.

La segunda herramienta utilizada para la realización de las validaciones fue Mobius [25] el cual establece un ambiente de verificación integrado con Eclipse, y basa sus verificaciones especialmente con ESC/Java2 (21). A pesar que esta herramienta está más "madura" y que informa que puede ser utilizada en versiones de Java iguales o superiores a 1.5, las pruebas realizadas con tipo genéricos no dieron resultados positivos, pues la herramienta no los soporta.

Las pruebas descritas en la sección 5 presentan formas alternativas de evaluar tanto las especificaciones con la implementación.

5. EVALUACIÓN DEL DESARROLLO FORMAL DEL "*CORE*" DE UNA RED SOCIAL CON JML

Uno de los aspectos finales y más importantes del proyecto es la evaluación y comparación de los resultados de la especificación e implementación realizada "manualmente" con posibles resultados generados de forma automática por herramientas diseñadas para tal fin, de tal manera que permita llegar a conclusiones sobre el uso y efectividad de esta herramientas, chequear si se ajustan a especificaciones e implementaciones como las desarrolladas en este proyecto, y si es posible establecer y proponer una metodología similar a esta herramientas, pero ajustada a los características del tipo de proyecto y del estilo de especificación y desarrollo utilizado.

En la etapa actual del proyecto se ha probado una herramienta de generación dinámica de Invariantes denominada *Daikon*, y se están realizando pruebas con dos herramientas asociadas con especificaciones algebraicas que permiten validar desarrollos realizados en Java, estas herramientas son *Heureka* y *Congu*.

5.1. DAIKON [22]. GENERACIÓN AUTOMÁTICA DE INVARIANTES.

Daikon es una herramienta que produce invariantes "probables" mediante la observación de las propiedades que se mantienen durante la ejecución de la aplicación [26]. Daikon chequea las entradas y salidas de los procedimientos, generando posibles invariantes que corresponden a pre y post-condiciones, es decir, evalúa las potenciales invariantes contra los valores observados en las rutinas centrándose en el estado interno de los métodos.

Daikon además de generar las "posibles" invariantes para una aplicación, permite llevar éstas a diferentes lenguajes, entre ellos JML, permitiendo de esta forma, comparar fácilmente invariantes generadas por la herramienta contra las especificadas "manualmente" en el desarrollo formal de una aplicación con JML. Daikon requiere para la obtención de un buen número de invariantes que se realice un conjunto de pruebas lo suficientemente representativo para que la herramienta haga una adecuada evaluación, sin embargo, en las pruebas realizadas con la implementación del "core" de la red social no se obtuvieron los mejores resultados.

5.1.1. Resultados de las pruebas con Daikon

La implementación realizada para el "*core*" de la red social se utilizó como fuente para generar "Invariante Dinámicas" con Daikon. Con el fin de generar archivos "trace" lo suficientemente robustos para que la herramienta pudiera obtener mejores invariantes, se prepararon varios test de la aplicación orientados a probar cada uno de los métodos de la implementación de "`SocialNetwork`".

La generación de los archivos "trace" se realiza corriendo el programa bajo el control del "Instrumenter" *`Chicory`* , quien adiciona instrucciones a los programas Java para generar archivos de salida con los valores de las variables. Las pruebas realizadas se ejecutaron con archivos de pruebas para "SocialNetwork" (`testSN`), generando un archivo "trace".

Una vez se crea el archivo "trace" se procede a ejecutar Daikon, buscando que la invariantes sean generadas en JML. En un principio las primeras pruebas se realizaron con la configuración default de Daikon.

Finalmente, se ejecutan las instrucciones que permiten incluir las invariantes generadas (Ej. `SocialNetwork.inv.gz`) en los archivos fuentes de la implementación.

En principio las invariantes "probables" generadas por Daikon no son las esperadas para la implementación del "*core*" de la red social. La siguiente es una muestra de los resultados obtenidos en esta primera generación.

```
....
/*@ invariant this.sf != null; */
/*@ invariant this.sf != null; */
/*@ invariant pk_socialNetwork.SocialFriends_r.friend != null; */
/*@ invariant pk_socialNetwork.SocialFriends_r.friend != null; */
/*@ invariant pk_socialNetwork.SocialFriends_r.friendship != null; */
/*@ invariant this.PERSON != null; */
/*@ invariant this.PERSON != null; */
/*@ invariant this.RAWCONTENT != null; */
/*@ invariant this.RAWCONTENT != null; */
.....

 /*@ requires rc != null; */
 /*@ requires rc.id != null; */
 /*@ requires rc.nameContent != null; */
 /*@ ensures this.PERSON == \old(this.sf.FRIEND); */
```

```
/*@ ensures rc.id != null; */
/*@ ensures rc.nameContent != null; */
public void make_visible_rc(RawContentSN rc, PersonSN pe) {

      visible.add(new RelationContent(pe,rc));    // visible := visible \/ {pe |-> rc}

}
```

Tanto las invariantes de clase, como las pre y postcondiciones de los métodos de la implementación, solo evalúan el estado de nulidad de la variables, sin tener en cuenta si los métodos modifican las variables o no, ni cuál debería ser el estado o comportamiento de las variables una vez salgan del método. También se observa que se incluyen invariantes redundantes dentro de las especificaciones de las clases y pre y postcondiciones "no interesantes", es decir, que no aportan lo suficiente para describir el comportamiento del método.

Con el fin de solucionar el problema anterior y generar más pruebas y valores en los archivos "trace", las nuevas pruebas realizadas se ejecutaron con modulos para "SocialNetwork" (`testSN`) y para "SocialFriends" (`testSF`), generando dos nuevos archivos "trace

La inclusión de las nuevas pruebas se complemento activando diferentes opciones en el archivo de configuración de Daikon, buscando obtener mayor cantidad de invariantes y "más interesantes". La siguiente es una muestra de la instrucción final utilizada con todas las opciones descritas:

- `java daikon.Daikon --format JML --files_from files_traces.txt -o SocialNetwork.inv.gz --no_text_output --omit_from_output rs --config configSN.txt`

Después de múltiples pruebas los resultados obtenidos por la herramienta siguen ser los esperados, a pesar que se mejoran algunos aspectos (se eliminan las redundancias), no se obtienen ni invariantes de clase, ni precondiciones, ni postcondiciones ajustas a la implementación realizada. Incluso, la adición de las pruebas de la clase "`SocialFriends`" lleva a la herramienta a generar una gran cantidad de invariantes y sentencias JML que en algunos casos no se ajustan a la realidad de los métodos establecidos en la implementación.

Las nuevas invariantes generadas por Daikon, se proporcionan especificaciones que no se requieren, se generan solo especificaciones de nulidad y la instrucción JML "*`modifies`*" incluye una cantidad de variables que realmente este método no modifica,

Las últimas pruebas realizadas sobre la herramienta nos llevan a concluir que Daikon no soporta variables de tipos HashSet y HashMap, las cuales son la utilizadas para definir la mayor cantidad de variables de la implementación del "*core*" de la red social.

Por los resultados obtenidos, llegamos a la conclusión que Daikon no es una herramienta adecuada para generar invariantes sobre el tipo de implementación realizada en este proyecto.

5.2. HEUREKA. "Descubriendo Especificaciones Algebraicas desde Clases Java" [27]

Heureka es una herramienta que permite realizar especificaciones algebraicas desde dos (2) puntos de vista diferentes. En primer lugar permite generar especificaciones algebraicas tomando como base clases Java. Este proceso inicia mapeando las clases Java a "signatures" algebraicas para al final llegar a establecer axiomas, que representen el comportamiento del sistema.

Una de las características principales de la herramienta es que permite "describir que implementan las clases Java sin revelar los detalles de la implementación", es decir, trabaja sobre especificaciones funcionales de alto nivel en los componentes del software. La Figura No. 8, muestra un ejemplo de las especificaciones generadas por Heureka.

TYPE $IntStack$
FUNCTIONS
$IntStack : \rightarrow IntStack \times void$
$push : IntStack \times int \rightarrow IntStack \times void$
$pop : IntStack \rightarrow IntStack \times int$
AXIOMS
$\forall s : IntStack, i : int$
$pop(push(s, i).state).retval = i$
$pop(push(s, i).state).state = s$
$pop(IntStack().state).retval \rightsquigarrow \text{ArrayIndexOutOfBoundsException}$

Fuente: Henkel, Johannes and Diwan, Amer. *Discovering Algebraic Specifications from Java Class*. University of Colorado at Boulder. 2003.

Figura 9. Ejemplo de especificaciones generadas por Heureka

El segundo objetivo de Heureka, es validar una implementación realizada en Java utilizando unas especificaciones Algebraicas dadas [28], sería posible entonces, generar especificaciones algebraicas desde la herramienta, modificarlas según los requerimientos reales del sistema y validar el sistema con estas especificaciones.

Heureka se desarrollo como parte de la tesis Doctoral de Johannes Henkel [29], sin embargo, la herramienta no ha tenido ninguna evolución, ni se generó una versión pública de la herramienta desde esa fecha.

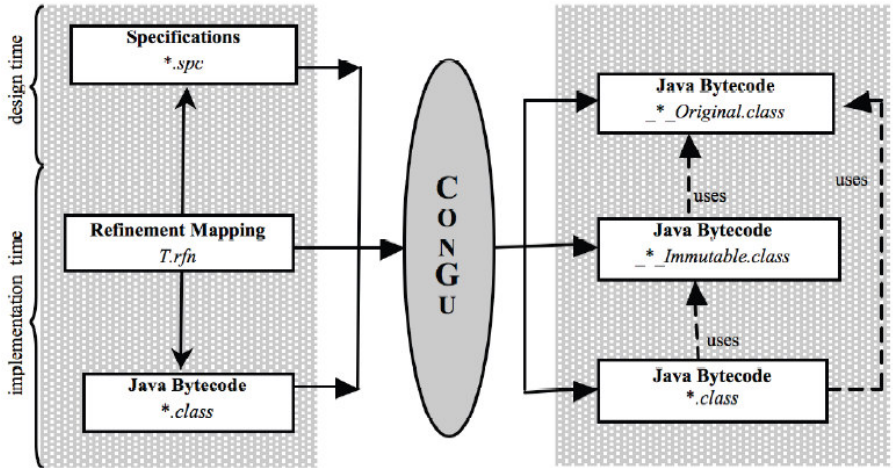
A pesar de lo anterior y teniendo en cuenta la importancia que podría tener para el desarrollo del proyecto, se obtuvo una versión de la herramienta y actualmente estamos en proceso de instalación y realización de pruebas.

La idea general es poder utilizar la herramienta para generar especificaciones algebraicas con base en la implementación desarrollada para la red social y buscar con esta misma herramienta u otra (Ej. ConGu) la verificación de la especificación e implementación desarrollada.

5.3. ConGu [30]. "Chequeando Clases Java Contra Especificaciones Algebraicas"

ConGu es una herramienta que permite chequear Clases Java contra propiedades dadas en especificaciones algebraicas. El chequeo consiste en determinar en tiempo de ejecución si las clases a analizar se comportan de acuerdo a la especificación. La herramienta requiere entonces, una especificación, unas clases escritas en Java y un mapeo (*refinement mapping*) que relaciona las clases y métodos con las especificaciones dadas.

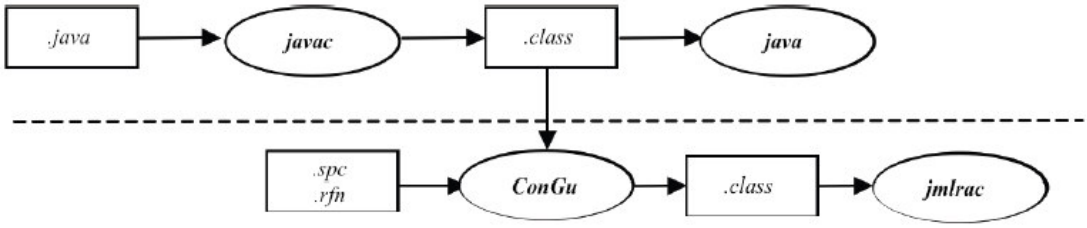
La idea general de la herramienta es poder generar automáticamente los "contratos" (en JML) de las especificaciones sobre las clases Java, a fin de verificar la ejecución de las clases que implementan las especificaciones. La Figura No. 9 representa el funcionamiento de ConGu.



Fuente: Vasconcelos, Nunes, Lopes, Ramiro and Crispim. *Monitoring Java Code Using ConGu*. Universidad de Lisboa. Junio 2008

Figura 10. Entradas y salidas en ConGu

El proceso de verificación y monitoreo de las aplicaciones Java con ConGu, inicia con la compilación de las clases Java (utilizando javac). Los archivos Java Bytecode obtenidos (*.class), las especificaciones algebraicas (*.spc) y los refinamientos de mapeo (.rfn) son las entradas para ConGu, el cual genera nuevos archivos Java modificados con "contratos" en JML de acuerdo a las especificaciones dadas, los cuales finalmente son verificados con la herramienta jmlrac, para chequear el comportamiento de la aplicación con relación al "contrato. La figura No. 10 muestra el flujo del proceso de verificación con ConGu.



Fuente: Vasconcelos, Nunes, Lopes, Ramiro and Crispim. *Monitoring Java Code Using ConGu*. Universidad de Lisboa. Junio 2008

Figura 11. Proceso de verificación con ConGu.

La idea general con esta herramienta es poder generar especificaciones JML dada una especificación algebraica (posiblemente generada por otra herramienta o realizada manualmente) y validarlas con las realizadas previamente en la especificación de la red social, con el fin de explorar el comportamiento de la herramienta, la validez de la metodología utilizada para este caso particular y la forma en que podría aportar a la conclusión del proyecto.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Actualmente nos encontramos en una etapa de exploración de diferentes herramientas que permitan validar de una u otra forma la especificación e implementación realizadas para el "core" de la red social. Un aspecto importante a evaluar en las pruebas con cada una de las herramientas seleccionadas es la metodología utilizada en el proceso de obtención, generación o validación de los "contratos" en JML, pues esta observación nos podría llevar a establecer un tipo comportamiento especifico para casos particulares como el desarrollado en el presente proyecto.

Esperamos también como conclusión de este proyecto, poder establecer una metodología que recoja los aspectos, enfoques, paradigmas y estilos de diseño utilizados en la fase de especificación, diseño e implementación del desarrollo formal del "core" de una red social. Será importante validar en consecuencia, si la metodología planteada solo es aplicable a desarrollos similares a los de una red social o es posible generalizarlo para otro tipo de desarrollos.

**** Faltaría Complementarla, pero es necesario terminar algunas pruebas que se están realizando actualmente *****

7. REFERENCIAS BIBLIOGRÁFICAS

1. **UM, Universal McCann.** *Wave-5. The Socialisation of Brands. Social Media Tracker.* 2010.

2. **Facebook.com.** [En línea] [Citado el: 3 de Noviembre de 2010.] http://www.facebook.com/press/info.php?statistics.

3. **ROMERO, Pablo.** El insaciable apetito de las redes sociales. *El mundo.com.* [En línea] 18 de Enero de 2008. [Citado el: 8 de Noviembre de 2010.] http://www.elmundo.es/navegante/2008/01/18/tecnologia/1200676010.html.

4. **MORALES, Raúl.** Las Redes sociales tienen serios problemas de seguridad. *Tendencias Informáticas. Universidad Politécnica de Madrid.* [En línea] 2008. [Citado el: 3 de 11 de 2010.] http://www.tendencias21.net/Las-redes-sociales-tienen-serios-problemas-de-seguridad_a2541.html.

5. **KORNBLUM, Janet and MARKLEIN, Mary Beth.** USA TODAY.com. *What you say online could haunt you.* [En línea] 3 de Agosto de 2006. [Citado el: 3 de Noviembre de 2010.] http://www.usatoday.com/tech/news/internetprivacy/2006-03-08-facebook-myspace_x.htm.

6. **BBC NEWS.** *Wife Murdered for Facebook status.* [En línea] 23 de Enero de 2009. [Citado el: 3 de Noviembre de 2010.] http://news.bbc.co.uk/2/hi/uk_news/england/staffordshire/7845946.stm.

7. National Center For Missing and Exploited Childen. [En línea] [Citado el: 3 de Noviembre de 2010.] http://www.missingkids.com.

8. **BOYD, Danah, and ELLISON, Nicole.** Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication.* [En línea] 2007. Vols. 13(1), article 11.

9. **Electronic Privacy Information Center.** [En línea] [Citado el: 3 de Noviembre de 2001.] http://epic.org/.

10. The Java Modeling Language. *JML Home Page.* [En línea] [Citado el: 3 de Noviembre de 2010.] http://www.eecs.ucf.edu/~leavens/JML/.

11. **ABRIAL, Jean-Raymond.** *The B-Book: Assigning programs to meanings.* s.l. : Cambridge University Press, 1996.

12. **CATAÑO, Nestor, Y RUEDA, Camilo.** Matelas: A Predicate Calculus Common Formal Definition for Social Networking. *Universidad de Madeira, Pontificia Universidad Javeriana-Cali.* [En línea] 2009. http://abzconference.org/.

13. **LEAVENS, Gary and CHEON, Yoonsik.** Design by Contract with JML. *Iowa State University and University of Texas at El Paso.* [En línea] Septiembre de 2006.

14. **CATAÑO, Néstor and WAHLS, Tim.** *A Baker Recipe to Port B Machines into JML Specifications.*

15. **SCHNEIDER, Steve.** *The Formal B-Methods. Series: Cornerstones of Computing* . s.l. : Palgrave Macmillan, 2001.

16. **ABRIAL, Jean-Raymond.** Guidelines to Formal System Studies. [En línea] 2000.

17. **MEYER, Bertrand.** *Object-Oriented Software Construction.* New Jersey : Prentice Hall PRT, 2007. Segunda Edición.

18. **LEAVENS, Gary, BAKER, Albert and RUBY, Clyde.** JML: a Notation for Detailed Design. Behavioral Specifications for Businesses and System. *Kluwer Academic Publishers.* s.l. : Capitulo 12. Págs. 175-188, 1999.

19. **BURDY, Lilian, LEAVENS, Gary, y otros.** An overview of JML tools and applications. Software Tools for Technology. [En línea]

20. **BREUNESSE, C.B., CATAÑO, Nestor, HUISMAN, M., and JACOBS, B.** Formal Methods for Smarts Cards: an experience report. *Elsevier Science.* [En línea] Febrero de 2008.

21. KindSoftware : ESC/Java2. [En línea] http://secure.ucd.ie/products/opensource/ESCJava2/.

22. The Daikon Invariant Detector. [En línea] http://groups.csail.mit.edu/pag/daikon/.

23. SourceForge - jmlspecs. [En línea] http://sourceforge.net/apps/trac/jmlspecs/wiki/OpenJml.

24. JML and MultiJava Documentation. [En línea] [Citado el: 3 de Noviembre de 2010.] http://www.eecs.ucf.edu/~leavens/JML-release/javadocs/org/jmlspecs/models/package-summary.html.

25. KindSoftware: Mobius Program Verification Environment. [En línea] http://secure.ucd.ie/products/opensource/Mobius/.

26. **ERNST, Michael D, y otros.** The Daikon system for dynamic detection of likely invariants. *Elsevier Science.* [En línea] Octubre de 2007.

27. **HENKEL, Johannes and DIWAN, Amer.** Discovering Algebraic Specifications from Java Class. *University of Colorado at Boulder.* [En línea] 2003.

28. —. A Tool for writing and Debugging Algebraic Specifications. *University of Colorado at Boulder.* [En línea] 2004.

29. **HENKEL, Johannes.** *Discovering and Debugging Algebraic Specifications for Java Clases. Tesis Doctoral.* s.l. : University of Colorado, 2004.

30. ConGu. Contract Guided System Development. [En línea] http://gloss.di.fc.ul.pt/congu.