Refactorizaciones de especialización en cuanto a genericidad. Definición para una familia de lenguajes y soporte basado en frameworks

Raúl Marticorena Sánchez[1], Carlos López Nozal[1] y Yania Crespo González-Carvajal[2]

[1] Universidad de Burgos, Dpto. de Ingeniería Civil
Área de Lenguajes y Sistemas Informáticos
{rmartico, clopezno}@ubu.es
[2] Universidad de Valladolid, Dpto. de Informática
{yania}@infor.uva.es

Resumen El presente trabajo presenta la definición de un conjunto de refactorizaciones de especialización sobre clases genéricas. Las refactorizaciones se definen en función de transformaciones sobre las características del lenguaje vinculadas con genericidad como: parámetros formales, parámetros reales, instanciaciones genéricas y variantes de acotación. El estudio de las refactorizaciones se presenta desde una cierta independencia del lenguaje, a través de su definición sobre un lenguaje modelo, lo cual permite especificar de manera independiente la refactorización para una gran familia de lenguajes. Dicho lenguaje modelo, así como la definición de las refactorizaciones, son soportados a través de una solución basada en frameworks. También se presenta un ejemplo de instanciación del framework con una extensión concreta de un lenguaje que soporta genericidad.
Palabras clave: refactorización, especialización, genericidad, tipos paramétricos, acotación, frameworks, lenguajes orientados a objeto.

1. Introducción

El actual interés de la programación genérica, dentro de la mayoría de lenguajes orientados a objetos, hacen de ésta un área relevante en la definición y aplicación de refactorizaciones. En particular la existencia de propuestas para su inclusión en lenguajes como Java [1] y en el .Net Common Language Runtime [2], en cuya especificación inicial no aparece, determinan la futura importancia de la definición de refactorizaciones en dicho ámbito.

Existen catálogos de refactorizaciones y diferentes definiciones de operaciones de refactorización que pueden considerarse de especialización o de generalización [3, 4]. Sin embargo en estos trabajos y catálogos no existen definiciones relacionadas con la genericidad. En este sentido, previamente en [5] se definió una

refactorización de generalización llamada **parameterize** mientras que en el presente trabajo se define una colección de refactorizaciones de especialización con genericidad.

Por otra parte, la definición de refactorizaciones de manera independiente del lenguaje [6, 5] ofrece una solución a las posibilidades de reutilización en el desarrollo de herramientas de refactorización cuando se adapten a nuevos lenguajes y nuevas operaciones de refactorización. En esta solución, el esfuerzo de la definición de refactorizaciones de manera general garantiza una recuperación del esfuerzo inicial y su aplicación futura en nuevos lenguajes.

En el establecimiento de una definición de un entorno para refactorizaciones independientes del lenguaje es necesario un soporte que permita representar las abstracciones de los distintos lenguajes de programación OO estáticamente tipados y con genericidad. Esta solución basada en frameworks [7] con representación de las características especificas de los lenguajes a través de extensiones de un conjunto de abstracciones definidas a partir del lenguaje modelo MOON[3] fue propuesta en [8]. En concreto se presentan las abstracciones definidas sobre MOON en lo referente a genericidad como una extensión del lenguaje GJ como propuesta de Java genérico.

En lo que sigue, el artículo se estructura de la siguiente forma: la Sección 2 presenta el conjunto de trabajos esencialmente relacionados con éste, la Sección 3 describe el proceso seguido en la definición de refactorizaciones de especialización sobre genericidad con independencia del lenguaje, para ello se describen una serie de predicados básicos para expresar pre y postcondiciones así como un conjunto de primitivas definidas sobre el lenguaje modelo MOON que especifican las operaciones realizadas en las refactorizaciones, la Sección 4 estudia el soporte de dichas refactorizaciones en una solución basada en frameworks y su instanciación concreta para un lenguaje como GJ [9], mientras que en la Sección 5 se concluye y se muestran las líneas de trabajo futuras.

2. Trabajos Relacionados

Existen trabajos previos en la definición de refactorizaciones independientes del lenguaje. En [6, 10] se presenta el modelo FAMIX como un meta-modelo para almacenar información incluyendo una herramienta para asistir a las refactorizaciones, denominada MOOSE [11]. Para la definición del modelo se centran en el estudio de dos lenguajes de características diferentes, como Smalltalk y Java. Al intentar dar respuesta tanto a lenguajes estática como dinámicamente tipados, no toman en cuenta características avanzadas en los lenguajes fuertemente tipados objeto de nuestro estudio, en particular sobre los tipos paramétricos y las clases genéricas.

Propuestas similares han sido realizadas en la definición de modelos para la descripción de lenguajes orientados a objetos bien a través de la definición del modelo OFL (Open Flexible Languages) [12] o con la propuesta de frameworks

[3] MOON es acrónimo de Minimal Object-Oriented Notation.

basados en una representación del software en forma de grafo, que soporten métricas con independencia del lenguaje [13]. Otra propuesta como [14] muestra la definición de transformaciones genéricas sobre diferentes paradigmas de programación a través de un framework.

En la definición de refactorizaciones nos encontramos con los trabajos iniciales de [3] con un planteamiento semiformal. Por otra parte existen también definiciones no formales a través de un catálogo en [4] y definiciones más formales [15] en base a precondiciones y postcondiciones, sustentadas en predicados.

En ninguno de los anteriores trabajos se plantean refactorizaciones relacionadas con la especialización de clases genéricas. Dicha situación puede venir motivada por la ausencia de dicha característica en lenguajes ampliamente extendidos como JAVA o C#, o bien se encontraba incluida pero su implementación no animaba a su uso como en C++. Sin embargo, la aparición de propuestas de inclusión [1, 2] de dicha característica en estos lenguajes impulsa a la definición de refactorizaciones en este contexto.

Hasta donde conocemos, éstos han sido los únicos trabajos que se han publicado en una línea similar, en mayor o menor medida, al nuestro: estudiar refactorizaciones de especialización sobre genericidad, con un alto grado de independencia del lenguaje y soportadas por una solución basada en frameworks.

3. Refactorizaciones de Especialización en Genericidad

Esta sección presenta la definición de refactorizaciones de especialización relacionadas con aspectos vinculados a la genericidad. Se entiende por especialización la eliminación de elementos estructurales que dificultan la comprensión y aumenten la complejidad del diseño, cuya eliminación no altera el comportamiento inicial de las clases en el contexto dado.

Para poder definir las refactorizaciones se ha seguido un proceso divido en tres fases: definición de predicados básicos sobre el lenguaje modelo MOON (Sección 3.1), definición de primitivas de edición que operan directamente modificando el código MOON (Sección 3.2), y por último, la definición de las refactorizaciones propiamente (Sección 3.3). En esta última fase se definirán los elementos de cada una de las refactorizaciones, siguiendo parte de la estructura ya planteada en [4] y [16] dando una definición completa de una de ellas (Sección 3.4). El objetivo final es la definición de cada refactorización mostrando su descripción, motivación y entradas, junto con una definición semiformal de las precondiciones, acciones y postcondiciones basada en los predicados y primitivas introducidos.

3.1. Definición de Predicados sobre MOON

Para la definición de refactorizaciones se describen una serie de predicados básicos que se utilizarán posteriormente en la definición de las precondiciones y postcondiciones a cumplir en las refactorizaciones definidas.

La información utilizada en la definición de predicados se extrae a partir de la información contenida en el código fuente (según MOON) de las clases de un

repositorio. La información de interés para el caso de las refactorizaciones que nos ocupan está relacionada con las declaraciones de parámetros formales de las clases, acotaciones sobre dichos parámetros y declaraciones de tipo en el cuerpo de las clases dependientes de dichos parámetros formales. MOON incluye tres variantes de acotación: subtipado [17, 18], conformidad [19] y cláusulas **where** [20, 21].

A continuación se presenta brevemente una clasificación de predicados, señalando como ejemplo alguno de los predicados presentes en la categoría. La definición de todos los predicados se puede consultar en [22].

Información de tipos e instanciación Describen los tipos parámetricos, completa e incompletamente instanciados, a partir de las declaraciones de clases que los implementan, información sobre las relaciones de herencia, así como la información de subtipado.
Ejemplos:
- *Desc(C)* siendo C una clase se define como el conjunto de clases de $\mathcal{C}$ (conjunto de clases) tal que existe un camino de herencia a C con origen en dichas clases.
- *IsCompleted(T)* siendo T un tipo el predicado se cumple cuando T es un tipo completamente instanciado.
- *Subtype(A)*, siendo A un tipo se define como el conjunto de tipos del repositorio que son subtipos válidos de A respetando las reglas de tipos de MOON que establecen obligatoriamente la existencia de una relación de herencia (directa o indirecta) entre sus clases determinantes y teniendo en cuenta las reglas de variantes (subtipado, conformidad y cláusulas **where**) definidas en MOON.

Información de parámetros genéricos y sustituciones Información sobre los parámetros genéricos asociados en las declaraciones de clases genéricas así como la información de tipos que dan sustitución a los mismos.
Ejemplo:
- *SubstFormalPar(C,G)*, siendo C una clase genérica y G un parámetro formal se define como el conjunto de sustituciones reales con tipos completos a G de la clase C.

Información de acotación Describen los distintos tipos de acotación en las tres variantes soportadas por MOON.
Ejemplo:
- *Bound_Type(C,G)*, siendo C una clase genérica y G un parámetro formal se define como el tipo que acota al parámetro formal G en la clase C, en las variantes de MOON de subtipado y conformidad.

3.2. Definición de Primitivas de Edición

En esta sección se presenta un conjunto de primitivas definidas sobre el lenguaje MOON, implicadas en la modificación de las clases genéricas. Teniendo en cuenta que dichas primitivas operan directamente a nivel sintáctico sobre la gramática definida en MOON [8].

Primitivas generales primitivas que operan con características básicas de las clases genéricas:

- *DeleteFormalParameter(C,G)* elimina el parámetro genérico formal G de una clase genérica C.
- *DeleteRealParameter(C,G,B)* elimina las sustituciones reales al parámetro genérico formal G en las instanciaciones de una clase genérica C en el cuerpo de una clase cliente B.
- *SubstituteFormalParameter(C,G,T)* sustituye el parámetro genérico formal G en el cuerpo de una clase genérica C por el tipo T.

Primitivas en acotación por subtipado y conformidad primitivas que operan en los tipos de acotación por subtipado y conformidad, de los parámetros formales:

- *ReplaceBoundType(C,G,T)* reemplaza la acotación de un parámetro genérico formal G por el tipo T en una clase genérica C.

Primitivas en acotación por cláusulas *where* primitivas que operan en la acotación por cláusulas **where**, de los parámetros formales:

- *AddWhereClause(C,G,W)* añade una nueva cláusula where W en la acotación del parámetro genérico formal G en una clase genérica C.
- *AddSignatureInWhereClause(C,G,F)* añade un nuevo tipo funcional F a la cláusula **where** en la acotación del parámetro genérico formal G en una clase genérica C.

3.3. Definición de Refactorizaciones de Especialización

Una vez definidos los predicados básicos y el conjunto básico de operaciones sobre clases genéricas, se puede definir el conjunto de refactorizaciones de especialización formado por:

- ***Specialize Bound S*** sustituir el tipo de la acotación por un subtipo.
- ***Specialize Bound F*** sustituir acotación F [23] por acotación por subtipado.
- ***Replace Formal Parameter with Type*** reemplazar un parámetro genérico formal por un tipo.
- ***Replace Formal Parameter with Type Bound*** reemplazar un parámetro genérico formal por el tipo de su acotación.
- ***Add Where Clause*** añadir una cláusula **where** como acotación de un parámetro genérico formal.
- ***Add Signature in Where Clause*** añadir una signatura de método en una cláusula **where**.

Dado que la variante de acotación por conformidad en MOON puede ser vista como un tipo particular de acotación por subtipado, las refactorizaciones **Specialize Bound S** , **Replace Formal Parameter with Type**, **Replace Formal Parameter with Type Bound** pueden aplicarse en el caso de la variante de acotación por conformidad en MOON, cambiando el concepto de subtipado por el de conformidad.

A continuación se describen cada una de ellas únicamente desde los aspectos de descripción y motivación de la refactorización. Por motivos de brevedad sólo se describe una refactorización en todos sus aspectos (descripción, motivación, entradas, precondiciones, acciones y postcondiciones) en la Sección 3.4 estando disponible una descripción completa del resto en [22].

Specialize Bound S La refactorización consiste en especializar el tipo de la acotación de un parámetro formal en una declaración de clase genérica por un subtipo (Figura 1). Esto es sólo aplicable en aquellos lenguajes que permitan acotación por subtipado.

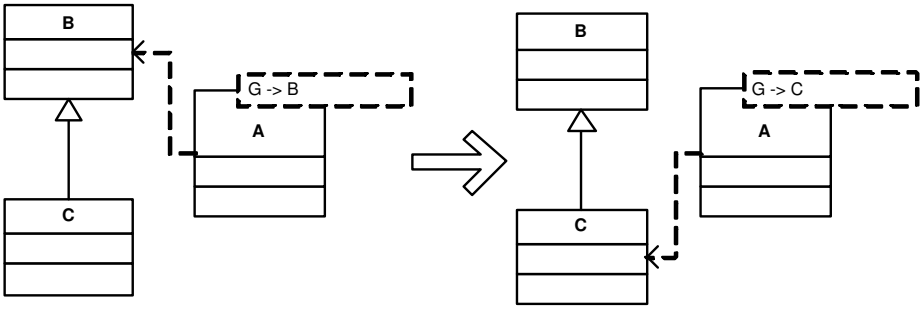

Figura 1. Specialize Bound S

Motivación En genericidad con acotación mediante subtipado, se limitan las instanciaciones genéricas válidas, a los subtipos del tipo con el que se acota y se restringe el conjunto de propiedades utilizadas a través de entidades declaradas con el parámetro genérico G de la acotación. En la práctica podemos encontrarnos que el conjunto de sustituciones reales de las instanciaciones genéricas se limita a un subconjunto de descendientes.

Si se elimina la actual acotación, y se sustituye por el ancestro común a las clases que se están utilizando como parámetros reales, se pierde genericidad, pero se gana claridad respecto al conjunto de sustituciones reales, así como la posibilidad futura de modificar el código con las propiedades intrínsecas definidas en el nuevo tipo de la acotación. Esta refactorización se podría aplicar a la hora de acotar parámetros no acotados inicialmente, en los que la cota es el tipo universal, permitiendo aplicarse esta transformación a lenguajes que inicialmente no incorporaban acotación.

Specialize Bound F La refactorización consiste en especializar el tipo de la acotación del parámetro formal en una declaración de clase genérica con acotación F, sustituyendo la declaración con acotación F por una acotación por subtipado, con un tipo completamente instanciado como acotación (Figura 2). Esto es sólo aplicable en aquellos lenguajes que permitan genericidad con acotación F.

Motivación En genericidad con acotación F, se puede encontrar el caso de que todas las instanciaciones genéricas toman como sustitución real un único tipo.

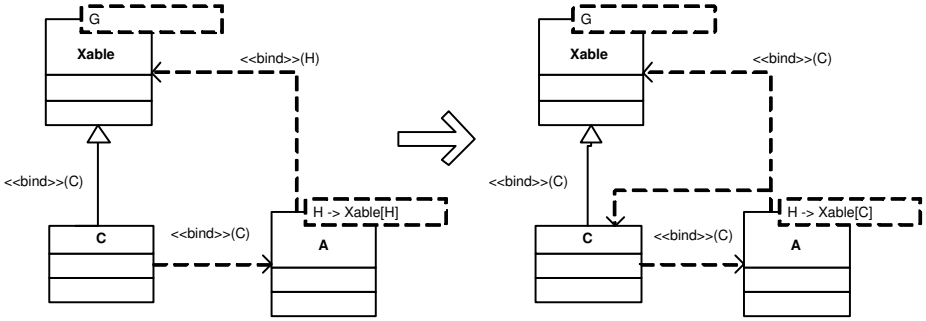

Figura 2. Specialize Bound F

En este caso podemos sustituir la acotación F por subtipado, simplificando la complejidad de la acotación F pasando al caso particular de acotación por subtipado. La refactorización puede ser especialmente útil en la transformación hacia un lenguaje que no soporte acotación F, pero sí soporte acotación por subtipado (e.g. EIFFEL).

Replace Formal Parameter with Type La refactorización consiste en eliminar un parámetro formal de la definición de una clase genérica, sustituyendo en el cuerpo de la clase el uso del parámetro formal por un tipo completamente instanciado (Figura 3).

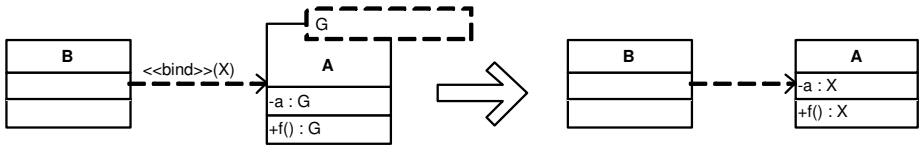

Figura 3. Replace Formal Parameter with Type

Motivación La refactorización surge cuando dada una clase genérica con acotación por subtipado, la sustitución real para uno de sus parámetros formales es siempre el mismo tipo.

Como solución se plantea la eliminación del parámetro formal en la declaración de la clase, común en todas las instanciaciones, sustituyendo el tipo genérico con el tipo completamente instanciado en el cuerpo de la clase genérica.

Esta refactorización afecta a los clientes directos y descendientes directos de la clase puesto que la declaración de tipo que referencia a la clase, cambia al eliminarse un parámetro genérico formal. Esto permite ver la diferencia entre las refactorizaciones que afectan siempre a varias clases, frente a las primitivas de edición que afectan a la estructura de una única clase y no tienen en cuenta bajo qué condiciones se pueden ejecutar para garantizar la corrección del código resultante y la preservación del comportamiento.

Replace Formal Parameter with Type Bound La refactorización consiste en eliminar un parámetro formal de la definición de una clase genérica, sustituyendo todo uso del parámetro formal como declaración de tipo por el tipo acotación de dicho parámetro (Figura 4). Como caso particular tenemos el caso de genericidad no restringida, donde la acotación es el tipo universal ***Object*** y se obraría de igual manera.

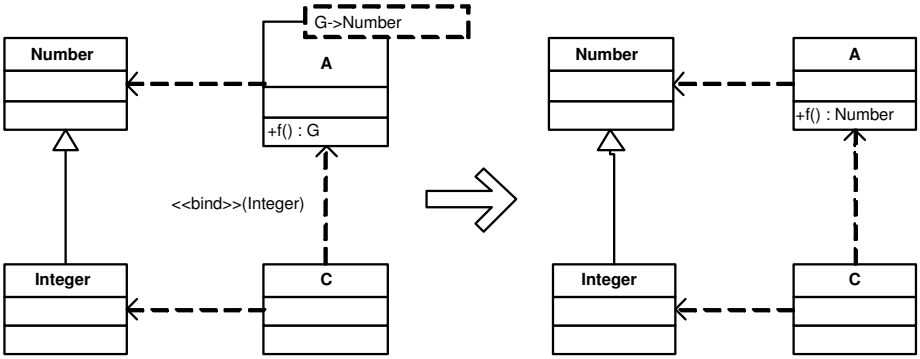

Figura 4. Replace Formal Parameter with Type Bound

Motivación La refactorización surge cuando dada una clase genérica con acotación se plantea la eliminación del parámetro genérico por el tipo acotación. Se consigue la eliminación de genericidad en la clase y una solución al problema para facilitar la integración de código legado no genérico (e.g. *raw types* en la propuesta GJ [9]).

Esta refactorización afecta a los clientes directos y descendientes directos de la clase puesto que la declaración de tipo que referencia a la clase, cambia al eliminarse un parámetro genérico formal. Además en esta refactorización, los clientes estarán obligados al uso de conversión de tipos (*downcast*), con el objetivo de la simulación del polimorfismo paramétrico con polimorfismo de inclusión.

Add Where Clause La refactorización consiste en añadir una nueva cláusula ***where*** como acotación a uno de los parámetros formales de la clase (Figura 5).

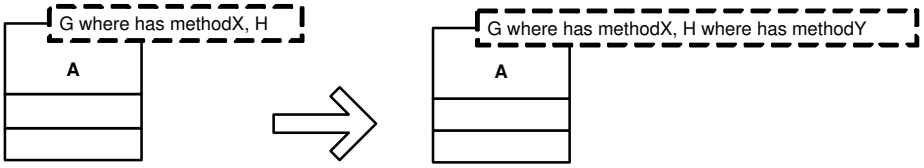

Figura 5. Add Where Clause

Motivación Cuando todas las clases dadas como sustitución al parámetro formal tienen en común un conjunto de signaturas, se puede añadir como acotación de dicho parámetro una cláusula **where** que agrupe el conjunto de signaturas comunes. Esto limita el conjunto de sustituciones reales a dicho parámetro es-

pecializando la clase, permitiendo el uso futuro de las características añadidas y mejorando la comprensión de la solución dada.

Add Signature in Where Clause La refactorización consiste en añadir a la acotación por cláusulas ***where*** de uno de los parámetros genéricos formales, una nueva signatura de método (Figura 6).

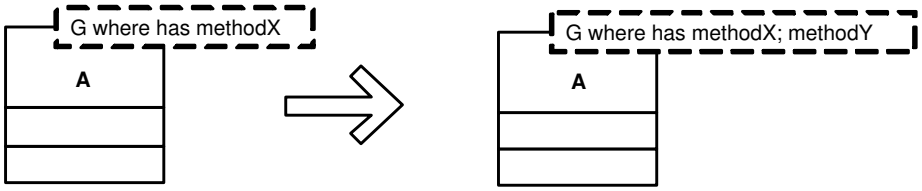

Figura 6. Add Signature in Where Clause

Motivación Cuando todas las clases dadas como sustitución a un parámetro formal tienen en común un método, se puede añadir a la acotación de dicho parámetro en la cláusula **where** dicho método. Esto limita el conjunto de sustituciones reales a dicho parámetro permitiendo el uso futuro de esa propiedad a través de las entidades cuya declaración de tipo es el parámetro formal.

3.4. Ejemplo de Definición en Base a Predicados y Primitivas

En esta sección se toma como caso de estudio la refactorización ***Specialize Bound S***, para mostrar la definición de la refactorización estableciendo su relación con los predicados y primitivas definidos anteriormente.

Los predicados son la base para definir las precondiciones y postcondiciones de la refactorización, mientras que las primitivas nos permiten definir las operaciones o acciones a realizar sobre el lenguaje modelo para llevar a cabo la refactorización. El cumplimiento de las precondiciones, y la ejecución de las acciones debe garantizar que es una refactorización correcta (preserva el comportamiento). Esta es una postcondición no escrita común a todas las refactorizaciones. La especificación completa de la refactorización se encuentra en la Tabla 1.

4. Independencia del Lenguaje: Solución Basada en Frameworks

Las abstracciones definidas en MOON, deben dar soporte a las características concretas de los distintos lenguajes, relevantes respecto a un conjunto de refactorizaciones definidas sobre ellas. En este sentido se pueden considerar las características de los distintos lenguajes de programación orientados a objetos como un dominio del conocimiento común a partir del cuál se pretende definir un conjunto de predicados y primitivas utilizadas en la definición de refactorizaciones. La solución que permita evolucionar el modelo conforme a una independencia del

Tabla 1. Refactorización ***Specialize Bound S***

Descripción	especializar el tipo de la acotación de un parámetro formal en una declaración de clase genérica por un subtipo
Motivación	en genericidad con acotación mediante subtipado, se limitan las instanciaciones genéricas válidas, a los subtipos del tipo con el que se acota y se restringe el conjunto de propiedades utilizadas a través de entidades declaradas con el parámetro genérico G de la acotación. En la práctica podemos encontrarnos que el conjunto de sustituciones reales de las instanciaciones genéricas se limita a un subconjunto de descendientes.
Entrada	clase genérica (C), parámetro genérico formal acotado (G) y nuevo tipo de la acotación (A).
Precondiciones	1. G es un identificador genérico formal en la clase C. $G \in FormalParam(C)$ 2. Todos las sustituciones completas al parámetro genérico son subtipo de A. $\forall X \in SubstFormalPar(C, G) \Rightarrow X \in Subtype(A)$ 3. El tipo A es un subtipo de la acotación del parámetro formal. $Bound\_Type(C, G) = C'/A \in Subtype(C')$ 4. La acotación del parámetro formal en los descendientes de las clase genérica C es un subtipo de A. $\forall C' \in Desc(C)/Bound\_Type(C', G) = T \wedge IsCompleted(T) \Rightarrow T \in Subtype(A)$
Acciones	1. $ReplaceBoundType(C, G, A)$
Postcondiciones	1. La acotación actual del parámetro G es A. $Bound\_Type(C, G) = A$

lenguaje debe dar soporte a estas características específicas además del conjunto de características comunes definidas en MOON.

Un framework es un diseño reutilizable de todo o parte de un sistema software descrito por un conjunto de clases abstractas y la forma en la que esas clases abstractas colaboran [24]. Bajo esta definición, se puede considerar el framework como un soporte que permite abordar el problema de la independencia del lenguaje. Uno de los elementos clave que aparecen en un framework son los puntos de extensión, que expresan las partes variables de las abstracciones del modelo, las características concretas de los lenguajes en nuestro caso. La variación se

alcanza mediante puntos de extensión de caja blanca, creando clases y métodos abstractos dando soporte a una funcionalidad predefinida de caja negra. La definición del framework, encapsula el conjunto de abstracciones con las características comunes de los lenguajes y un conjunto de puntos de extensión para expresar las específicas. La definición de las refactorizaciones con independencia del lenguaje, basada en predicados y primitivas de edición ilustrada en la sección anterior tiene su reflejo sobre las propiedades de las abstracciones del framework MOON.

4.1. Abstracciones Referentes a Genericidad en el Framework MOON

Al soportar MOON genericidad los tipos son clasificados en parámetros formales (FORMAL_PAR) y tipos implementados por las propias definiciones de las clases (CLASS_TYPE). Cuando la definición de una clase no es genérica la asociación entre la clase (CLASS_DEF) y el tipo (CLASS_TYPE) es única, es decir se considera a una clase como una construcción lingüística en un LOO que se usa para implementar los tipos. Por tanto el tipo vendrá dado directamente por la definición de la clase. Cuando la definición de una clase es genérica, se puede decir que es una clase determinante de un conjunto potencialmente infinito de tipos donde cada una de las instanciaciones genéricas realizadas se corresponde con distintos tipos (CLASS_TYPE).

La definición de una clase genérica consta de una lista de parámetros formales. En las variantes de acotación en MOON se persigue acotar las características de los parámetros formales para garantizar la corrección de tipos en las instanciaciones genéricas determinando un conjunto de sustituciones válidas para los parámetros formales. Si un parámetro formal no está acotado, el tipo del parámetro real en una sustitución está acotado por el tipo universal ***Object*** contenido en MOON. Dentro del conjunto de tipos obtenidos a partir de las instanciaciones genéricas se distinguen dos subconjuntos:

- Los tipos completos o completamente instanciados, procedentes de instanciaciones genéricas completas, son aquellos cuyo conjunto de parámetros reales no contiene ningún parámetro formal, es decir, el conjunto de parámetros reales permanece fijo.
- Los tipos no completos o incompletamente instanciados, procedentes de instanciaciones genéricas no completas, cuyo conjunto de parámetros reales contienen al menos un parámetro formal, es decir, el conjunto de parámetros reales permanece variable dependiendo del contexto.

En la Figura 7 se muestra un diagrama de clases que representan las abstracciones de MOON, sus relaciones y un conjunto de invariantes definidos sobre la definición de la clase (CLASS_DEF) y sobre los tipos procedentes de las definiciones de las clases (CLASS_TYPE).

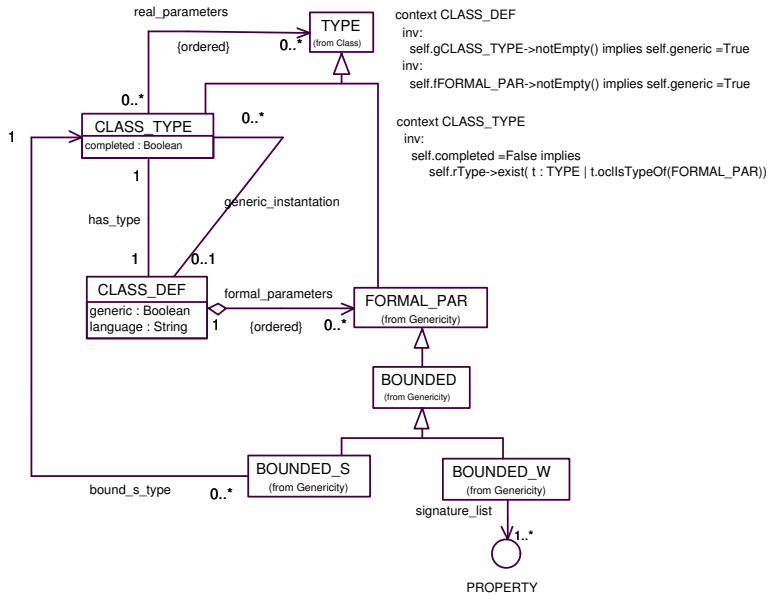

Figura 7. Definición de clases genéricas en MOON soportando acotación por subtipado (o conformidad) y acotación por cláusulas **where**

4.2. Extensión de MOON para GJ: Acotación F en GJ

En este apartado se presenta GJ como una propuesta para definir tipos genéricos en JAVA a través de una extensión del lenguaje [1, 9]. Además se presenta el estudio de la acotación F como una característica específica del lenguaje de programación GJ que sirve como ejemplo de extensión del framework de MOON.

El objetivo perseguido es poder capturar la información contenida en códigos fuente escritos en GJ sobre las abstracciones definidas en MOON y poder aplicar las refactorizaciones de especialización en genericidad definidas en la sección anterior. Por ser GJ un lenguaje de programación orientado a objetos con genericidad, las abstracciones básicas del lenguaje están contenidas en MOON, pero existen características concretas del lenguaje que no están soportadas directamente. A partir del ejemplo de definición de refactorización dado en la Tabla 1 se mostrará una relación con las clases identificadas en el framework propuesto.

La acotación F [23] es una variante de la acotación por subtipado cuando se trabaja con tipos mutuamente recursivos en el conjunto de parámetros formales de la clase genérica que los contiene. Un tipo recursivo es aquel cuya especificación esta definida en términos del propio tipo. Las reglas sintácticas del lenguaje modelo MOON soportan directamente la acotación F, por abordar la acotación

de subtipado y los tipos procedentes de instanciaciones genéricas no completas. Sin embargo a la hora de dar soporte sobre el framework, se debe reflejar la semántica de la coexistencia de ambos tipos de acotaciones, subtipado y F. Para dar soporte a esta nueva característica se añade al modelo de la Figura 7 dos puntos de extensión sobre la clase BOUNDED_S :

- isBoundedF():Boolean, que sirve para interrogar si la acotación por subtipado es una acotación F.
- getRecursiveFormalPar():FORMAL_PAR, que permite obtener el parámetro formal recursivo que forma parte de la acotación.

Además se añade una restricción: *"Si el parámetro formal está acotado por la variante F entonces el CLASS_TYPE obtenido a través de la relación bound_s_type no está completamente instanciado"*. Esto se puede ver en la Figura 8.

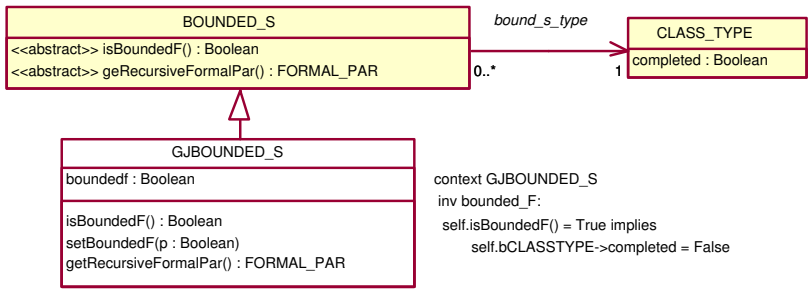

Figura 8. Extensión acotación F para GJ

4.3. Relación de Abstracciones del Framework con la Definición de Refactorizaciones

La definición de las refactorizaciones que se presentó en la Sección 3 se basa en las reglas sintácticas del lenguaje MOON para definir tanto los predicados como las primitivas de edición. Es necesario que estas operaciones puedan reflejarse en las clases que conforma el framework MOON. A continuación se muestra una relación de las clases del framework respecto a los predicados y primitivas de edición de la refactorización tomando como ejemplo ***Specialize Bound S***.

Predicados (Pre/Postcondiciones):

- **FormalParam(C)**: con CLASS_DEF y FORMAL_PAR.
- **SubstFormalPar(C,G)**: con CLASS_DEF, FORMAL_PAR, CLASS_TYPE y real_parameters.
- **Subtype(C)**: con CLASS_DEF, INHERITANCE_CLAUSE y CLASS_TYPE.
- **Bound_Type(C,G)**: con CLASS_DEF, FORMAL_PAR y BOUNDED_S.
- **Desc(C)**: con CLASS_DEF, INHERITANCE_CLAUSE y CLASS_TYPE.

- **IsCompleted(T)**: con CLASS_TYPE.

Primitivas de Edición:

- **ReplaceBoundType(C,G,T)**: con CLASS_DEF, FORMAL_PAR, BOUNDED_S y CLASS_TYPE.

5. Conclusiones y Líneas de Trabajo Futuro

El presente trabajo establece una base para la definición de un conjunto de refactorizaciones de especialización en lo referente a tipos paramétricos y clases genéricas. Ante el vacío actual, hasta donde conocen los autores del presente documento, de líneas de trabajo en esta dirección, y teniendo en cuenta las posibilidades futuras, se vislumbra la definición de refactorizaciones de este tipo como un campo de gran interés.

Si además, dichas refactorizaciones pueden ser definidas y soportadas sobre un framework que soporte un lenguaje modelo como MOON, que cubre una amplia familia de lenguajes, y la posibilidad de extender dicho framework a instanciaciones concretas para cada lenguaje, se abre el camino a la posibilidad de reutilización del esfuerzo de definición de refactorizaciones.

Respecto a la independencia del lenguaje se hace necesario ampliar el estudio de las características que puedan afectar a las refactorizaciones. Para ello se propone definir extensiones sobre el estudio de nuevos lenguajes.

La actual propuesta nos conduce a trabajar en la línea de la definición de nuevas refactorizaciones así como la elaboración final de una herramienta que permita el soporte de las mismas a través de una solución basada en frameworks.

Referencias

1. Gilad Bracha, Norman Cohen, Christian Kemper, Steve Marx, Martin Odersky, Sven-Eric Panitz, David Stoutamire, Kresten Thorup, and Philip Wadler. Adding generics to the java programming language: Participant draft specification, 2001.
2. Andrew Kennedy and Don Syme. Design and implementation of generics for the .net common language runtime. In ACM, editor, *ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*. Microsoft Research, ACM, 2001.
3. William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, Urbana-Champaign, IL, USA, 1992.
4. Martin Fowler. *Refactoring. Improving the Design of Existing Code*. Addison Wesley, 2000.
5. Yania Crespo. *Incremento del potencial de reutilización del software mediante refactorizaciones*. PhD thesis, 2000. Disponible en http://giro.infor.uva.es/docpub/crespo-phd.ps.
6. Sander Tichelaar, Stéphane Ducasse, Serge Demeyer, and Oscar Nierstrasz. A meta-model for language-independent refactoring. In *Proceedings ISPSE 2000*, pages 157–167. IEEE, 2000.

7. Mohamed Fayad, Goug Schmidt, and Ralph Johnson. *Building Applications Frameworks: Object-oriented Foundations of Framework Design*. John Wiley & Sons, 1999.
8. Y. Crespo, V. Cardeñoso, and J.M. Marqués. Un lenguaje modelo para la definición y análisis de refactorizaciones. In *Actas PROLE'01, Almagro, España*, November 2001. Disponible en http://giro.infor.uva.es/docpub/crespo-prole2001.pdf.
9. Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Gj: Specification, 1998.
10. Sander Tichelaar. *Modeling Object-Oriented Software for Reverse Engineering and Refactoring*. PhD thesis, 2001.
11. Stéphane Ducasse, Michele Lanza, and Sander Tichelaar. Moose: an extensible language-independent environment for reengineering object-oriented systems. In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET 2000)*, 2000.
12. Pierre Crescenzo and Philippe Lahire. Customisation of inheritance. In *Inheritance Workshop at ECOOP 2002*, pages 23–29. Black et al. Eds, Information Technology Research Institute, Jyvaskyla University Press, 2002.
13. Tom Mens and Michele Lanza. A graph-based metamodel for object-oriented software metrics. In Tom Mens, Andy Schrr, and Gabriele Taentzer, editors, *Electronic Notes in Theoretical Computer Science*, volume 72. Elsevier, 2002.
14. Ralf Lämmel. Towards Generic Refactoring. In *Proc. of Third ACM SIGPLAN Workshop on Rule-Based Programming RULE'02*, Pittsburgh, USA, October5 2002. ACM Press. 14 pages.
15. Donald Bradley Roberts. *Practical Analysis for Refactoring*. PhD thesis, 1999.
16. Lance Tokuda and Don S. Batory. Evolving object-oriented designs with refactorings. In *Automated Software Engineering*, 1999.
17. Luca Cardelli. A semantics of multiple inheritance. *Semantics of Data Types*, LNCS(173):51 – 68, 1984.
18. Luca Cardelli and Peter Wegner. On understanding types, data abstraction and polymorphism. *Computing Surveys*, 17(4):471 – 523, 1985.
19. Bertrand Meyer. *Object-Oriented Software Construction*. McGraw Hill, 2nd edition, 1997.
20. Barbara Liskov. Programming methodology group progress report. Technical report, Laboratory for Computer Science Progress Report XIV, MIT Laboratory for Computer Science, 1977.
21. Barbara Liskov, Dorothy Curtis, Mark Day, and Sanjay Ghemawat. *Theta Reference Manual*. Programming Methodology group Memo 88. MIT Laboratory for Computer, 1995.
22. Raúl Marticorena. Refactorizaciones de especialización sobre el lenguaje modelo MOON. Technical Report DI-2003-02, Departamento de Informática. Universidad de Valladolid, septiembre 2003. Disponible en http://giro.infor.uva.es/docpub/marticorena-tr2003-02.pdf.
23. Peter S. Canning, William Cook, Walter L. Hill, John C. Mitchell, and Walter G. Olthoff. F-bounded quantification for object-oriented programming. In *Proceedings of the ACM Conference on Functional Programming and Computer Architecture*, pages 273–280, september 1989.
24. Don Roberts and Ralph E. Johnson. Evolving frameworks: A pattern language for developing object-oriented frameworks. In *Pattern Languages of Program Design 3*. Addison Wesley, 1997.