Framework de Evaluación para Modelos Formales de Patrones de Diseño

Andrés Pablo Flores
Departamento de Ciencias de la Computación
Universidad Nacional del Comahue
Bs.As. 1400 – 8300 – Neuquén
E-mail: aflores@uncoma.edu.ar

Pablo Ruben Fillottrani
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Av. Alem 1253 – 8000 – Bahía Blanca
E-mail: prf@cs.uns.edu.ar

Abstract
Patrones de diseño es una herramienta de soporte como técnica de reuso y anticipación al cambio. Su descripción es habitualmente bastante informal no permitiendo concluir cuándo y cómo un patrón es aplicado correctamente. En esto el uso de lenguajes formales ayuda a su descripción. En este trabajo se resumen las limitaciones de los lenguajes informales y los requisitos de un lenguaje formal para la adecuada descripción y utilización de patrones de diseño. Dadas las diferentes opciones en modelos formales actuales, se conformó un Framework de Evaluación compuesto de dos partes. La primera incluye 14 características para analizar los lenguajes formales utilizados para desarrollar los Modelos de Patrones OO. La segunda se compone de 10 aspectos referentes a los modelos propiamente dichos. El objetivo de este Framework es adquirir mayor conocimiento de las ventajas y falencias de los modelos formales y así comprender aún mejor los elementos constituyentes requeridos en un lenguaje formal para una apropiada representación de patrones. Los diseñadores pueden beneficiarse con un proceso de Diseño basado en Patrones soportado por una descripción precisa de patrones con la flexibilidad adicional de herramientas automáticas basadas en los modelos formales desarrollados.

Palabras Claves: Ingeniería de Software, Métodos Formales, Patrones de Diseño

1 Introducción

La eficiencia en el proceso de diseño puede ser perfeccionada para evitar el consumo excesivo de tiempo y esfuerzo que sólo se transcribe a un desarrollo más costoso. La reusabilidad es un paso importante en esta dirección, que es cubierto por Patrones de Diseño. Cada patrón describe una solución genérica a un problema que es comúnmente encontrado en diferentes contextos, y resume la experiencia probadas de diversos diseñadores [1,2,3]. Los patrones de diseño se han hecho populares con la publicación del catálogo GoF [4] que introduce un vocabulario de comunicación durante un desarrollo de software de experiencias en problemas de diseño y sus posibles soluciones. La notación gráfico/textual utilizada para describir los patrones aporta un muy buen cuadro intuitivo de los mismos, pero no es lo suficientemente precisa como para permitir que un diseñador demuestre cuándo y cómo un problema específico tiene correspondencia con un patrón particular. Además la descripción posee un grado intencional de abstracción para así capturar un amplio rango de aplicabilidad. Es por lo tanto, extremadamente difícil proveer una certificación significativa de correctitud para aquel software que ha sido desarrollado usando patrones [5]. Mediante una notación más precisa se puede ayudar a mejorar el entendimiento acerca de los patrones y evitar ambigüedades e inconsistencias, las cuales son inherentes en la descripción gráfico/textual.
Al profundizar en estos aspectos se puede observar que es ampliamente reconocido que el uso de un lenguaje formal puede proveer el medio para una conveniente descripción de patrones. Algunos de los trabajos además de suministrar especificaciones de algún conjunto de patrones incluso han provisto de la definición de nuevas notaciones. Algunos de los modelos fueron construidos mediante fórmulas de LePUS [12,13]; o a través de VDM++ [16]; o por medio de extensiones de UML [20,27]. La intención de estos enfoques es optimizar el desarrollo de nuevos sistemas o bien mejorar

sistemas previos con el uso de patrones de diseño y posiblemente el ajuste apropiado del sistema luego de la aplicación de los mismos. Sin importar la intención, desarrollo o refactoring, la necesidad de descripción formal de los patrones se comprende perfectamente. Nuestro grupo también ha llevado a cabo un desarrollo formal, el cual consta de una base formal para diseño OO y la aplicación de patrones de diseño. Nuestra premisa ha sido alcanzar un apropiado Diseño basado en Patrones, para el cual se ha especificado formalmente usando RSL (El lenguaje de especificación RAISE [6]) las propiedades de un diseño orientado a objetos general en el cual es posible establecer la aplicación de patrones de diseño [22,7,9,10,11]. Nuestro enfoque de Diseño basado en Patrones se ajusta a una clasificación Bottom-Up [8]. Una versión previa formalizada mediante el mismo lenguaje especifica los componentes de los patrones de forma más directa, de manera que el modelo formal puede ser considerado un enfoque Top-Down [5].

Con la intención de analizar los actuales esfuerzos en representaciones formales de patrones de diseño orientados a objetos, presentamos en este articulo un Framework de Evaluación compuesto de dos partes principales. Una de ellas se conforma de características relevantes acerca de los modelos de patrones de diseño que pudieran ser puestos bajo evaluación. La otra parte se compone de propiedades de los lenguajes formales que hubieren sido aplicados para la descripción de aquellos enfoques. El Framework será ilustrado mediante el análisis de 7 modelos y lenguajes; y se brindará una descripción detallada solo de algunas características. El análisis completo puede verse en [33]. La meta de este Framework de Evaluación es ganar mayor conocimiento acerca de las ventajas de las descripciones formales de patrones de diseño. Al resaltar las mejores características de cada modelo y el lenguaje formal utilizado se puede mejorar el entendimiento de los elementos constituyentes requeridos que un lenguaje formal debería incluir para una apropiada representación de patrones de diseño. Los diseñadores pueden verse beneficiados a través de un proceso de diseño soportado por una conveniente descripción de patrones. Este proceso puede ser mejorado aún más mediante el suministro de herramientas automáticas basadas en los modelos formales desarrollados. Así el principal objetivo es alcanzar un proceso de Diseño basado en Patrones con suficiente flexibilidad y adecuada precisión para proyectos tanto de desarrollo como de refactoring.

El articulo está organizado como sigue. A continuación se presenta el fundamento de realización de este trabajo, las características que restringen la expresividad de los lenguajes informales. La sección 3 introduce el framework de evaluación completo, ilustrado con algunos ejemplos. Las conclusiones de este trabajo son presentadas en la sección 4.

2 Limitaciones de los Métodos Informales

Las notaciones de objetos son inadecuadas como lenguajes de especificación de patrones de diseño, si se considera que ninguno ha sido diseñado para describir patrones. En cada una de las notaciones comunes [23,24], sus los símbolos designan constructores concretos en Lenguajes de Programación Orientados a Objetos (OOPL). El ejemplo típico es OMT [25], usado en la sección *Estructura* de cada patrón, en el catálogo GoF [4]. Como lenguajes de especificación de patrones de diseño, las notaciones de objetos son aún menos expresivas que los fragmentos de programas mostrados en las secciones de *Ejemplos de Código* de cada patrón. La razón es el desarrollo de las mismas con la carencia de una fuerte semántica formal [12]. El objetivo principal es proveer un lenguaje de especificación que asegure la calidad de un diseño para reducir las posibilidades de errores durante el desarrollo debido a imprecisión o mala interpretación. A continuación vemos un resumen de las falencias de los lenguajes informales:

PRECISIÓN

Que la especificación desarrollada no esté abierta a múltiples interpretaciones. Restringir la libertad de interpretación se puede incrementar las posibilidades de una mejor comprensión y la reducción en la introducción de errores. En particular las situaciones que llevan a la imprecisión son:

Ambigüedad: El contenido de la especificación puede ser interpretado de muchas formas distintas. En los lenguajes de especificación de patrones se observa la necesidad de agregar lenguaje natural para aclarar cierta información. El resultado es el anexo de sentencias que traen consigo la problemática de ambigüedad, la cual es sensible al contexto. Es decir, dependiendo del entorno en el cual se aplica un patrón se le dará un significado u otro, y posiblemente también al diagrama de clases propuesto como solución genérica.

Incompletitud y Vaguedad: La especificación informal no documenta adecuadamente ciertas situaciones, las cuales pueden conducir a una aplicación incorrecta bien sea por la descripción confusa o indefinida de un aspecto importante, la ausencia total de éste, o bien por el agregado erróneo de aspectos indeseables. Esto además es un factor que influye sobre el desarrollo de herramientas de soporte, que necesitan ser provistas de información exacta, consistente e integral.

Inconsistencia: A lo largo de una especificación es a veces factible descubrir que los elementos descriptos manifiesten contradicciones entre ellos. Esto es muy común en el lenguaje informal dado que no presenta posibilidad de ser verificado (probado o demostrado) adecuadamente. En la descripción de patrones es común descubrir que ciertos aspectos, a la luz de un contexto particular, se vuelven inconsistentes.

ESPECIFICACIÓN DE ESTRUCTURAS DE PATRONES

El lenguaje de especificación de patrones debería proveer de las herramientas necesarias para describir la estructura de los patrones de forma que no se incurra en aspectos indeseables como los mencionados previamente. Dos características son de interés a este respecto:

Relaciones Insubstanciales: Muchos lenguajes de especificación carecen de la capacidad necesaria para especificar relaciones significativas entre las entidades involucradas. Las relaciones entre los participantes no solamente involucran un aspecto sintáctico sino también semántico, dado que facilitan la descripción del comportamiento de cooperación que sucede entre las entidades. Así un lenguaje pobre o carente de semántica no será capaz de detallar colaboraciones entre entidades, lo cual es fundamental en el paradigma orientado a objetos.

Insuficiencia de Abstracción: La capacidad de representar generalizaciones en orientación a objetos es un aspecto que no puede ser relegado dado que provee uno de los mecanismos de reuso que posibilitan la flexibilización de un proceso de desarrollo. Además en conjunto con la composición conforman el fundamento para que los patrones puedan describir aspectos que pueden variar en un diseño. Sin embargo muchos lenguajes presentan inhabilidad para capturar abstracciones, proveyendo algún mecanismo que carece de algunas de las características necesarias, o bien simplemente no soportando las generalizaciones.

Algunas características que deberían mantenerse en un enfoque de formalización de patrones son:

Beneficio de Patrones: Que la especificación si bien no mejore, que al menos no entorpezca la aplicación de patrones. Es decir, el objetivo de la formalización es subsanar los aspectos antes mencionados, pero en el intento de proveer una mejora no se debería modificar la semántica ampliamente reconocida de un patrón. De esta manera la aplicación de patrones en lugar de verse empeorada por el uso de formalidad, será extensamente beneficiada al identificarse de forma precisa el momento oportuno y el modo correcto de su aplicación.

Condición de ser OO: Algunos lenguajes formales utilizados para especificar patrones no fueron originalmente creados con tal finalidad. Esto hace que la representación de aspectos de orientación a objetos pueda no estar completamente soportada por el lenguaje. Así en algunos enfoques de formalización se deben proveer analogías para establecer el respaldo de ciertas características de objetos. Como se explicó en el ítem anterior la formalización debería al menos no inyectar mayor imprecisión en la especificación. Esto también se relaciona con los aspectos de orientación a objetos en los cuales se basa la teoría de patrones.

3 Framework de Evaluación de Lenguajes y Modelos Formales para Patrones OO.

Las características que conforman el Framework de Evaluación han sido identificadas de numerosos trabajos aportados a la comunidad científica y que se reconocen como de gran interés en el entorno de la Ingeniería de Software y particularmente de la aplicación de propiedades lógicas. El Framework de Evaluación no solo presta utilidad en el análisis de Modelos Formales de Patrones OO, sino también de los lenguajes que se hubieren aplicado. Al iniciar el estudio de los modelos se descubrió la necesidad de analizar también los lenguajes formales empleados dado que varios aspectos de los modelos son consecuencias de las propiedades inherentes a esos lenguajes. Así en la sección 3.1 se analizarán 7 lenguajes mediante la primera parte del Framework de Evaluación, la cual consta de 14 características. Esto facilitará el entendimiento sobre el análisis posterior de los modelos de patrones en particular. Luego en la sección 3.2 se presentará el análisis, bajo 10 características, de 9 modelos desarrollados bajo los lenguajes formales seleccionados. En la Tabla 1 se presenta el Framework de Evaluación, en el cual se establecen las valoraciones posibles para las características que serán analizadas. La descripción de cada característica se presenta en las secciones correspondientes a cada uno de los dos análisis principales.

<table>
<tr><th>Característica</th><th>Valoración</th><th>Característica</th><th>Valoración</th></tr>
<tr><td colspan="2">Lenguajes Formales</td><td>13-Representación Multi-Capa</td><td>No / Pobre – Muy Bueno</td></tr>
<tr><td>01-Fundamento Matemático</td><td>Lógica Matemática</td><td>14-Repositorio de Especificaciones</td><td>Pobre – Muy Bueno</td></tr>
<tr><td>02-Notación Visual</td><td>Si / No</td><td colspan="2">Modelos Formales de Patrones OO</td></tr>
<tr><td>03-Especificación Formal</td><td>Estilo Especificación</td><td>01-Precisión del Modelo de Patrones</td><td rowspan="2">Pobre – Muy Bueno</td></tr>
<tr><td>04-Especificación Estructural</td><td rowspan="3">* – *****</td><td>02-Completitud de Características de Patrones</td></tr>
<tr><td>05-Especificación de Comportamiento</td><td>03-Tipo de Modelo</td><td>Top-Down / Bottom-Up/ Mixed</td></tr>
<tr><td>06-Características OO</td><td>04-Heurísticas de Diseño</td><td rowspan="5">Pobre – Muy Bueno</td></tr>
<tr><td>07-Identificar Refinamientos</td><td rowspan="5">Pobre – Muy Bueno</td><td>05-Flexibilidad del Modelo</td></tr>
<tr><td>08-Especificar Restricciones</td><td>06-Implementación a Nivel de Diseño</td></tr>
<tr><td>09-Implementación a Nivel de Diseño</td><td>07-Extensibilidad</td></tr>
<tr><td>10-Comprensión</td><td>08-Facilidad de Uso</td></tr>
<tr><td>11-Extensibilidad</td><td>09-Diseño basado en Patrones</td><td>Si – No</td></tr>
<tr><td>12-Facilidad de Uso</td><td>* – *****</td><td>10-Administración de Repositorio de Patrones</td><td>Pobre – Muy Bueno</td></tr>
</table>

Tabla 1: Framework de Evaluación de Lenguajes y Modelos Formales de Patrones OO

3.1 Evaluación de Métodos Formales

Dadas las características analizadas en la sección 2 y como se puede observar mediante el análisis bajo el Framework de Evaluación que se presenta en este artículo, se podría concordar que la meta de un lenguaje de especificación de patrones podría ser aquel que incluya las propiedades que se presentan a continuación. Un lenguaje que a priori tiene la meta de conformar a estas propiedades es LePUS, el cual como se observa en [33] no presenta en particular características de comportamiento [19].

- Formal: Su semántica está bien definida y facilita el razonamiento del uso de un calculus bien conocido.
- Concisa: Está compuesto de un conjunto compacto de símbolos.

- Parsimonia: Introduce un número mínimo de adornos nuevos (con respecto a notaciones existentes)
- Expresividad: Adecuadamente abstracto; es decir compuesto de símbolos que puedan convenientemente expresar la esencia de tantos patrones de diseño como sea posible.
- Genérico: Soportar sentencias generales sobre programas y constructores; de un "alto nivel de genericidad", de manera que pueda ser complemento para el uso bajo una terminología arquitectural.
- Estructural y de Comportamiento, para que sea ameno para análisis y especificación estática y a la vez factible de describir la interacción dinámica de entidades.
- Gráfico, a través de una notación visual.

A continuación se presenta una descripción resumida de las características de la primera parte del Framework de Evaluación: para Lenguajes Formales. Los lenguajes formales que se han tomado para proveer un análisis comparativo han sido utilizados en trabajos que han provisto una representación precisa de patrones de diseño. Algunos de ellos son lenguajes de propósito general (e.g. RAISE), otros en cambio fueron desarrollados bajo un propósito específico (e.g. DisCo). Entre estos últimos, algunos han sido desarrollados con la intención de proveer un lenguaje preciso para especificar patrones de diseño (e.g. LePUS). Detalles sobre cada uno de los lenguajes seleccionados puede verse en [33]. La Tabla 2 resume el análisis completo de los lenguajes formales.

01- Fundamento Matemático
El leguaje de especificación debe estar sostenido por un buen conjunto de teorías matemáticas, suficientemente amplio para brindar un buen poder expresivo, y a la vez no demasiado extenso de manera que habilite aspectos contradictorios. Así lo que se pretende es que el modelo subyacente sea sano y completo, que posea una buena capacidad de abstracción y que su expresividad sea adecuada para el área en el cual se aplicará.

Ejemplo: La tabla 2 resume el fundamento matemático de cada lenguaje.

02- Notación Visual
Los mecanismos visuales aportan cierta facilidad para representar abstracciones del entorno. Esta descripción puede servir para expresar los primeros bocetos del escenario de interés, la cual se puede lograr de manera quizá más flexible que con una representación textual. Se debe garantizar que todo elemento de la notación visual esté soportado por una pieza de lenguaje formal. Así la especificación visual surgida de esta notación tendrá su equivalente en una especificación basada en sentencias formales textuales.

Ejemplo: Uno de los lenguajes con notación visual en [20], provee una variación de la representación de una clase UML, describiendo mediante conjuntos las instancias abstractas de la entidad. Esto corresponde a una variación de UML que utiliza Diagramas de Restricción [30] (en vez de OCL) para mejorar la precisión de las especificaciones.

03- Especificación Formal
Al conformar una especificación por medio de un lenguaje formal se puede tener la impresión de sentirse limitado por el aparato formal provisto. Esta limitación puede ser tanto en falta de expresividad debido a demasiada rigurosidad como a una insuficiencia de la misma que no provee garantías de exactitud. El instrumental provisto por el lenguaje formal debería ser apropiadamente flexible de manera que sea factible una representación que cubra todos los aspectos de interés y bajo un proceso de especificación que resulte agradable (no tedioso).

Ejemplo: Como se puede apreciar en la tabla 1 cada lenguaje formal en estudio puede ser clasificado según alguno de los estilos posibles [15]. Algunos de ellos cubriendo más de una categoría (e.g. RAISE), presentando un estilo orientado a propiedad (e.g. LePUS, DisCo), o bien adecuándose a un

estilo basado en modelo (e.g. VDM++).

04- Especificación Estructural
Se necesita la capacidad para detallar los aspectos estáticos de la orientación a objetos y particularmente de los patrones de diseño. Mucha información es descripta estáticamente por medio de diversos elementos de diseño como clases, atributos y métodos (interface) y las relaciones entre las clases participantes. Esta información brinda la capacidad de abstraer aquellos aspectos que necesitan ser puntualizados en una primera vista y que se mantienen al continuar con un agregado de mayor detalle.

Ejemplo: DisCo [17] y Contracts [26] por haber sido desarrollados para una especificación con una orientación diferente no pueden representar las estructuras convencionales. Esto sin embargo no impide la representación, bajo un ligero cambio en la perspectiva hacia los sistemas.

05- Especificación de Comportamiento
Los aspectos de comportamiento nos brindan mayor detalle de los escenarios que se suceden en el entorno de interés. Este comportamiento puede ser descripto por medios estáticos los cuales si bien presentan un buen cuadro de abstracción, generalmente carecen del poder expresivo para representar todos los aspectos que concurren bajo condiciones de dinamismo.

Ejemplo: RAISE [6] no provee de forma directa un modo específico (determinado) de representar aspectos dinámicos aunque su aparato formal permite fácilmente la descripción de estos aspectos bajo una adecuada rigurosidad.

06- Características OO
Dado que el interés primario de este trabajo es el análisis de los modelos formales de patrones de diseño, los lenguajes formales utilizados deberían permitir una adecuada representación de los elementos y aspectos de la orientación a objetos. Si bien algunos lenguajes no han sido creados con la intención de ser catalogados como orientados a objetos, pueden proporcionar los mecanismos para su descripción. De esta manera mucho de las características inherentes a los patrones pueden ser convenientemente explicadas y descriptas.

Ejemplo: DisCo [17] y Contracts [26] al haber sido desarrollados bajo una orientación diferente, no proveen todos los mecanismos OO de forma directa, sino bajo un ligero cambio de perspectiva. Por ejemplo en DisCo en lugar de representar entidades que presentan una interface (servicios o requerimientos que puede atender), describe cambios en el estado de las entidades mediante acciones aplicadas sobre ellas. Esto se asemeja a un modelo funcional más que OO.

07- Identificar Refinamientos
Un aspecto que se observa comúnmente en cualquier entorno es la multiplicidad de ocurrencias de clasificación. Esta característica permite efectuar generalizaciones o abstracciones sobre situaciones del mundo real. Además esta propiedad brinda un buen mecanismo de reuso de aquello ya conocido y registrado. Sin embargo se debe proveer de buenos instrumentos para particularizar o refinar el caso de estudio. En orientación a objetos la generalización y especialización proveen de poderosos dispositivos de reuso estático o estructural que facilitan el modelado de un dominio en estudio.

Ejemplo: Lenguajes como LePUS [19], VDM++ [16], $\varsigma+\rho$-Calculus [21,29] y UML [24] proveen de un mecanismo de refinamiento que daría soporte a una relación conceptual *Is-A*. En particular $\varsigma+\rho$-Calculus y LePUS son los únicos que describen exclusivamente relaciones Is-A, donde LePUS la implementa por medio de *herencia de interfaces* (conformando subtyping) bajo la premisa de "*programar para la interface*" (fundamento de la teoría de patrones). Por otro lado VDM++ y UML proveen de herencia sin restricciones a priori, es decir subclassing.

08- Especificar Restricciones
Se pretende una representación que establezca las condiciones bajo las cuales se presentará un escenario o los requisitos que se deben cumplir antes, durante, y después de que un evento se active. Un lenguaje formal generalmente suministra una solución adecuada a este respecto. Sin embargo cada uno presenta una capacidad heterogénea lo cual habla también acerca de su expresividad.

Ejemplo: Todos los lenguajes analizados proveen de un medio de describir restricciones, aunque podríamos destacar Contracts [26], LePUS [19] y RAISE [6] como los que más herramientas y flexibilidad aportan en este sentido.

09- Implementación a Nivel de Diseño
Sin que el lenguaje formal pierda las características benéficas de abstracción que lo diferencian de un lenguaje de programación, la descripción con un suficiente grado de detalle puede ser apropiada mediante analogías a nivel de programación (por ejemplo pseupo-código). Es decir, capacidad de representar la funcionalidad interna significativa de un comportamiento visible.

Ejemplo: Lenguajes como VDM++ [16], UML [24] y Contracts [26], proveen de adecuados mecanismos de descripción de detalles de implementación a nivel de diseño, dado que se proveen representaciones de constructores a nivel de OOPL. Esta capacidad se ha desarrollado por la necesidad de formalizar mediante un lenguaje de estilo rayando lo imperativo.

10- Comprensión
La lógica subyacente en el modelo del lenguaje formal no debería ser demasiado compleja si se aspira a que el lenguaje sea ampliamente aplicable. De similar manera las especificaciones generadas a partir de tal lenguaje formal no deberían ser difíciles de entender o interpretar para que ésta se convierta en un modo de comunicar ideas de forma precisa.

Ejemplo: Los lenguajes imperativos son generalmente más fáciles de entender como se observa con VDM++ [16] y UML [24]. Aún LePUS [19] (orientado a propiedad) se puede considerar bastante comprensible desde los modelos visuales, aunque no tanto desde las fórmulas vinculadas a los mismos.

11- Extensibilidad
En un entorno cambiante es necesario estar alerta a la necesidad de representar nuevos eventos o características que pudieran avecinarse. Bajo un método formal que permita efectuar agregados a las especificaciones se está en presencia de la posibilidad de no omitir estas variaciones. Esta propiedad posibilita además que se aplique una importante técnica de diseño, *el diseño para el cambio*, el cual se basa en el principio de *anticipación al cambio*.

Ejemplo: Se ha encontrado que LePUS [19] pareciera no ofrecer la posibilidad de agregar funcionalidad sin alterar los modelos preexistentes. Cada modelo, una vez establecido parece no proveer de puntos de flexibilidad en los cuales introducir nuevas características que no afecten la especificación original. Esto quizá se deba a la intención en el desarrollo del lenguaje de modelar patrones de diseño y no su aplicación en un contexto real.

12- Facilidad de Uso
No solo se está abordando la facilidad de comprensión y de descripción, sino que además sería propicio que el lenguaje facilite la capacidad de aprehensión de los elementos formales provistos. Cuando el aparato formal es muy complejo o demasiado amplio se puede observar que el área de uso se reduce a un pequeño grupo (más fácil de recordar). Esto podría dificultar el modelado acabado del caso de estudio y afectar la agilidad de su desarrollo, perjudicando así la descriptibilidad y la productividad.

Ejemplo: Quizá el lenguaje con menor facilidad de uso sea $\varsigma+\rho$-Calculus [21,29], lo cual se debe al calculus del lenguaje. Lo que más se ve empañado es la comprensión de los modelos. La curva de aprendizaje para un lenguaje como éste podría ser algo pronunciada, aún más si se considera que se debe alcanzar una adecuada productividad dentro de cualquier equipo de desarrollo.

N° Ca	DisCo	LePUS	RAISE	VDM++	$\varsigma+\rho$-Calculus	UML + Restricciones	Contracts
01	Base Lógica y Mat. fuerte para espec. interac. de sistemas reactivos. (TLA)	Modelo Mat. muy sano. Fórmulas en lógica monádica de alto orden. (HOML)	Fundamento riguroso basado en VDM-SL y otros lenguajes (ASL, ML, OBJ, Larch,etc).	Fundamento rigur. basado en VDM-SL. Mat. discreta, teoría de conj. y relac. (lógica formal subyacente)	Análogía OO de *Lambda (ë)-Calculus.* Conjunto de teorías sana y completa.	Fundamento matemático no muy riguroso, pero su teoría de conjuntos es bastante útil.	Especificación basada en invariantes
02	No. Pero se provee de una herramienta de animación de espec.s.	Si. Muy precisa. Algo engorrosa y no muy comprensible	No	No	No	Diagramas Visuales. Un enfoque es puramente visual	No
03	Especificación flexible, bastante Imperativa.	Fórmulas rigurosas, espec. basada en predicado.	Leng. amplio espectro, orient. a propiedad y a modelo.	Espec. orientada a modelo. Muy útil para formalizar OO.	Especificación Imperativa OO.	Espec. Imperat. UML-OCL con notas para restricción.	Especificación imperativa precisa de CC.
04	**	*****	****	*****	****	*****	**
05	*****	***	****	*****	***	***	*****
06	***	****	***	*****	****	*****	***
07	Bueno	Bueno	Bueno	Bueno	Bueno	Bueno	Regular
08	Pobre	Bueno	Muy Bueno	Regular	Regular	Regular	Muy Bueno
09	Bueno	Regular	Bueno	Muy Bueno	Bueno	Muy Bueno	Muy Bueno
10	Regular	Muy Bueno	Bueno	Muy Bueno	Pobre	Muy Bueno	Bueno
11	Bueno	Limitada	Bueno	Bueno	Bueno	Bueno	Bueno
12	***	****	***	****	**	*****	***
13	Bueno	No	Muy Bueno	No	No	Rep. en 3 niveles	Bueno
14	Bueno	Bueno	Bueno	Bueno	Pobre	Bueno	Pobre

Tabla 2: Análisis de Características en Lenguajes Formales

13- Representación Multi-Capa

Dependiendo de la etapa de desarrollo y de la intención detrás del proceso de especificación se puede esperar que un lenguaje formal permita una descripción a un alto nivel de abstracción o a niveles más concretos. Algunos lenguajes poseen estas facilidades, incluso interconectando modelos axiomáticos con otros más imperativos, o bien proveyendo la capacidad para producir especificaciones híbridas.

Ejemplo: De los lenguajes en estudio, se podría concluir que RAISE [6] es el único que posee la peculiaridad de cubrir diferentes estilos de especificación para proveer representación a diferentes niveles de abstracción. Aún en un mismo estilo se pueden refinar fácilmente las especificaciones mediante un método particular que se ha conformado para dar soporte a este proceso de desarrollo.

14- Repositorio de Especificaciones

Bajo la premisa de desarrollar componentes altamente cohesivos que puedan ser fácilmente integrados a nuevas especificaciones, se espera que el entorno para el lenguaje formal posea capacidad para almacenaje y administración de las especificaciones producidas en desarrollos previos. De esta manera los próximos proyectos se beneficiarán de las facilidades de reuso.

Ejemplo: En el caso de VDM++ [16] las herramientas provistas son bastante completas y amigables, y en continuo mejoramiento. La compañía IFAD A/S (*http://www.ifad.dk/*) es quien se ha orientado a estos desarrollos y a la utilización de VDM++ en diferentes proyectos de software.

3.2 Evaluación de Modelos Formales de Patrones OO.

El uso de la técnica de patrones de diseño conforma lo que se ha denominado Proceso de Diseño basado en Patrones, el cual genera diseños flexibles capaces de soportar modificaciones surgidas de requerimientos fluctuantes, ya sea cambios de forma o bien extensiones. El diseño basado en patrones involucra el enlace de elementos de patrones a elementos de dominio de diseño del usuario. Así un subconjunto de clases y relaciones en un diseño concilian a una especificación de un patrón, si sus propiedades son las mismas que aquellas de sus contrapartes en el patrón. Existen tres enfoques para realizar esta correspondencia que conforman la instanciación de patrones. En [8,10]. se puede observar una figura con un esquema de dos niveles de enlace diseño-patrón.

- Top-down: dada una descripción de un patrón, generar los componentes apropiados a nivel de diseño del usuario.
- Bottom-up: dado un (subconjunto de un) diseño del usuario y un patrón, realizar el enlace o verificar si un patrón se corresponde con éste.
- Mixed: dada una estructura a nivel de diseño del usuario que solo parcialmente se ajusta a una descripción de un patrón, generar los componentes faltantes del patrón a nivel de diseño del usuario.

A continuación se describen las características de análisis de Modelos Formales de Patrones OO de la segunda parte del Framework de Evaluación. Cada una será ilustrada con un ejemplo surgido del análisis que ha sido llevado a cabo de 9 modelos, el cual puede ser visto en [33]. Los modelos han sido seleccionados por sus propiedades singulares que son propias o debidas al lenguaje que se usó para su desarrollo. La tabla 2 resume el análisis completo.

01- Precisión del Modelo de Patrones
Se pretende que el modelo no modifique la semántica de los patrones presentado en sus catálogos de origen, sino por el contrario que mejoren la descripción de los mismos, expresándolos de una manera consistente y sin ambigüedades.

Ejemplo: El uso de DisCo [14] requiere un cambio en la perspectiva sobre los patrones. Por lo tanto ya desde el uso de este lenguaje se tiene una primera sensación de modificación. Por otro lado se puede observar que los patrones modelados se han especificado con un grado mayor de abstracción que el expresado en el catalogo GoF [4] que en ciertos casos puede ser perjudicial.

02- Completitud de Características de Patrones
Dado que los patrones están descriptos con un cierto grado de abstracción para permitir su amplia aplicabilidad, muchos de ellos sufren de carencias en su descripción. La descripción formal debería expresar claramente cuándo y cómo se debe hacer el agregado de los aspectos faltantes. Esto ayudaría a mejorar la aplicabilidad de los patrones al no percibir una dificultad de discernimiento durante el diseño basado en patrones.

Ejemplo: El modelo en LePUS resulta una descripción bastante completa, aunque presenta todas las propiedades sólo desde una perspectiva estática. Esto último también es una característica del modelo Bottom-Up en RAISE, aunque éste se reconoce como el más completo y con mayor documentación sobre posibles variaciones de patrones, bajo una consideración de flexibilidad.

03- Tipo de Modelo

Cada uno de los modelos pueden ser clasificados de acuerdo a la forma como se instanciarán los patrones: Top-Down, Bottom-Up, Mixed. Lo cual fue expresado al comienzo de esta sección.

Ejemplo: Solo el modelo en UML-OCL [27] y uno de los modelos en RAISE [7,22] se ajustan a una clasificación Bottom-Up . El modelo UML y Diagramas de Restricción no explica como se haría la instanciación de patrones. El nivel de clases debería ser un meta-nivel si se quiere que se ajuste a una clasificación Bottom-Up, pero no hay suficientes detalles que indiquen este aspecto.

04- Heurísticas de Diseño

El proceso de desarrollo se ve favorecido con la aplicación de principios y heurísticas de diseño. La arquitectura del sistema resultante puede exhibir en una buena medida cualidades fundamentales que podrían no haberse logrado de otra manera. El modelo de patrones debería proveer restricciones como tales dado que ayudan a mejorar el entendimiento sobre los patrones y el diseño en general.

Ejemplo: Los únicos modelos con mejor nivel en éste aspecto son aquellos desarrollados bajo RAISE. Los modelos fueron construidos bajo la premisa de lograr el mejor entendimiento sobre patrones y en esto los principios y heurísticas de diseño asumen un rol preponderante. Bajo los modelos en RAISE, una aplicación debe primero satisfacer las propiedades de diseño para luego satisfacer aquellas relacionadas con el patrón que se hubiere aplicado. Así un diseñador no solo puede aprender acerca del adecuado uso de patrones, sino también cuáles son los requisitos mínimos y generales para que su diseño sea considerado de una calidad moderada.

05- Flexibilidad del Modelo

Existen diversas formas de expresar un patrón en las cuales aún se respeta su intención original. Los catálogos de patrones expresan informalmente la posibilidad de manejar una cierta variación en la solución al aplicar un determinado patrón. El modelo formal de patrones debería permitir estas variaciones, las cuales no solo favorecen la aplicabilidad de los patrones sino que además restringen la posibilidad de expresar soluciones que se alejan de la intención de los mismos.

Ejemplo: El modelo en LePUS tiene un cierto grado de flexibilidad. Se describen jerarquías sin representar roles de subclases (cuando es oportuno), que permite el agregado de tantas clases como se necesite. Esto es similar al modelo en RAISE, con clases intermedias. En LePUS, el icono de jerarquía por ejemplo asegura que las interfaces serán implementadas en algún momento, lo cual en RAISE debió ser especificado explícitamente. Por ello se puede apreciar que la descripción en LePUS es más reducida que la de RAISE.

06- Implementación a Nivel de Diseño

Los patrones expresan parte de las colaboraciones de sus participantes por medio de *anotaciones* que se adjuntan a las signaturas de los métodos. Éstas expresan por medio de pseudo-código la parte de implementación de los métodos que no debería ser omitida para no desdibujar el comportamiento de los participantes. Sería conveniente que el enfoque de formalización de patrones permita describir al menos el comportamiento predefinido, aunque la posibilidad de agregar además comportamiento específico de un contexto también es una buena característica.

Ejemplo: Para los modelos en VDM++ como UML no se tiene demasiada documentación que indique en qué grado es factible el agregado de detalles de implementación mediante el uso de los modelos desarrollados. Sin embargo se puede inferir que dado el lenguaje empleado se podría tener un buen nivel en este aspecto.

07- Extensibilidad

Una propiedad importante es la capacidad de agregar especificaciones de patrones nuevos o bien

extender las propiedades establecidas para los patrones en las formalizaciones existentes. Esto hace que los modelos formales de patrones puedan ser actualizables a medida que se manipulan las especificaciones y se logra un aprendizaje mas acabado con su utilización.

Ejemplo: El modelo que se observa menos flexible en el agregado de nuevas características es aquel desarrollado en LePUS [18]. La explicación surge de lo que se discutió en el análisis de este lenguaje (ver sección 3.1).

08- Facilidad de Uso

Esta propiedad está relacionada con aspectos como la rapidez con que se comprende el modelo, la habilidad necesaria para su utilización, y la medida en que es necesario un conocimiento profundo de la lógica subyacente.

Ejemplo: De los modelos en UML, quizá aquel que posee el esquema de tres niveles y Diagramas de Restricción, aparenta ser algo más engorroso que el otro (UML-OCL). Esto ha sido reconocido por los mismos autores del modelo en [20].

09- Diseño basado en Patrones

Dado que el proceso de diseño suele conllevar un cierto grado de dificultad, se intenta proveer una mejora mediante el uso de patrones. Para lograr un buen proceso de diseño basado en patrones se requiere que el tipo del modelo se ajuste otra clasificación en lugar de Top-Down. Esta última solo ayuda a conformar un patrón particular, aunque la necesidad yace en la aplicación de éste en un diseño. Aplicar un patrón tiene sus consecuencias como se explica en el catálogo de GoF[4], por ello es necesario saber si se están usando adecuadamente para conocer el resultado de su aplicación. Lo que se plantea es poder establecer una visión globalizada o macro del diseño completo en que se aplica el patrón y no solo la vista particular y aislada del patrón.

Ejemplo: Según el análisis que se hizo en Tipo de Modelo, se puede concluir que aquellos que cumplen con este aspecto son los modelos Bottom-Up basados en RAISE [7,22] y UML-OCL[27]. El resto de los modelos utilizan algún otro mecanismo para lograr que se realice la aplicación de patrones, pero los modelos propiamente dichos no presentan esta posibilidad por sí mismos.

10- Administración de Repositorio de Patrones

No importa el objetivo para el cual se utilicen los patrones, ya sea nuevos diseños o re-ingeniería, siempre se trata de especificar un modelo formal para luego efectuar una instanciación de los mismos. Para ello se necesita proveer de un repositorio para las especificaciones formales de patrones, enmarcadas en un sistema que posea facilidades para su administración.

Ejemplo: En el caso del modelo Bottom-Up en RAISE se está desarrollando una herramienta basada en el modelo formal para la aplicación de patrones y la verificación de correctitud del diseño y de los patrones [31,32]. En el caso del modelo bajo VDM++ se está desarrollando una herramienta para refactoring: REFORM [16].

4 Conclusiones

Dada la falencia de los lenguajes informales para desarrollar especificaciones precisas, es ampliamente aceptado que el uso de una metodología formal provee el medio conveniente para el logro de mayor precisión. La aplicación de técnicas de reuso y anticipación al cambio requiere usar patrones de diseño como herramienta de soporte. Su descripción se hace comúnmente bajo términos informales que no permiten concluir cúando y cómo un patrón es aplicado correctamente. Por ello el uso de lenguajes formales ayuda a su descripción. En este trabajo se han presentado (sección 2) las limitaciones de los lenguajes informales y los requisitos que debería satisfacer un lenguaje formal para la adecuada descripción y utilización de patrones de diseño. Algunos de los trabajos que

intentan proveer una descripción formal de patrones, además de obtener especificaciones de algún conjunto de patrones incluso han provisto de la definición de nuevas notaciones. Se han rescatado 9 trabajos que expresan diferentes opciones en modelos formales. Por ello se llevó a cabo un análisis comparativo de los mismos con el objeto de identificar falencias y ventajas en cada uno de ellos. Para realizar el análisis se conformó un Framework de Evaluación compuesto de dos partes (sección 3). La primera incluye 14 características para analizar los lenguajes formales utilizados para desarrollar los modelos de Patrones OO (sección 3.1). La segunda compuesta de 10 aspectos de interés referentes a los modelos propiamente dichos (sección 3.2). El objetivo del análisis mediante este Framework de Evaluación es ganar mayor conocimiento acerca de las ventajas de los modelos formales. Al resaltar las mejores características de cada modelo y los lenguajes formales utilizados se puede mejorar el entendimiento de los elementos constituyentes que un lenguaje formal debería incluir para una apropiada representación de patrones. Los diseñadores pueden verse aliviados a través de un proceso de diseño soportado por una conveniente descripción formal de patrones. Este proceso puede ser mejorado aún más mediante el suministro de herramientas automáticas basadas en los modelos formales desarrollados. Así el objetivo principal es el logro de un proceso de Diseño basado en Patrones con suficiente flexibilidad y adecuada precisión tanto para desarrollo como para refactoring.

N° Ca	DisCo	LePUS	RAISE	VDM++	$\varsigma+\rho$-Calculus	UML + Restricciones	Contracts
01	Regular	Muy Bueno	Muy Bueno	Bueno	Bueno	Bueno	Regular
02	Regular	Bueno	Muy Bueno	Bueno	Bueno	Bueno	Regular
03	Top-Down	Top-Down	Col-Dinám.: Top-Down Estático Puro: Bottom-Up	Top-Down	Top-Down	OCL: Bottom-Up Diag.Restr: Top-Down	Top-Down
04	Pobre	Regular	Muy Bueno	Pobre	Pobre	Pobre	Pobre
05	Pobre	Bueno	Muy Bueno	Pobre	Pobre	Muy Bueno	Pobre
06	Pobre	Pobre	Bueno	Bueno	Pobre	Bueno	Pobre
07	Bueno	Regular	Bueno	Bueno	Bueno	Bueno	Bueno
08	Bueno	Muy Bueno	Bueno	Bueno	Regular	Muy Bueno	Bueno
09	No	No	Si: Bottom-Up	No	No	Si: UML-OCL	No
10	Regular	Bueno	Bueno	Bueno	Pobre	Bueno	Pobre

Tabla 3: Análisis de Características en Modelos Formales de Patrones OO

5 Bibliografía

1. Doug Lea. *Pattern-Discution FAQ*, "http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.hmtl", December 1999.
2. Zubeck Robert. *Much Ado About Patterns.* "http://www.acm.org/crossroads/xrds5-*1/patterns.html*", March 2000.
3. Appleton Brad. *Patterns and Software: Essential Concepts and Terminology*, Object Magazine Online 2(5). "http://www.enteract.com/~bradapp", November 1997.
4. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
5. Alejandra Cechich and Richard Moore. *A Formal Basis for Object-Oriented Patterns.* Proceedings of the 6th Asia-Pacific Software Engineering Conference: APSEC'99, Takamatsu, Japan, pg. 248-291, December 1999.
6. The RAISE Language Group. *The RAISE Specification Language*. BCS Practitioner Series. Prentice Hall, 1992.
7. Andres Flores, Luis Reynoso and Richard Moore. *A Formal Model of Object-Oriented Design and GoF Design Patterns.* Proc. of FME 2001, Berlín, Germany, LNCS 2021, Springer Verlag 2001(223-241), March 2001. From Technical Report 200, UNU/IIST, P.O. Box 3058, Macau, July 2000.
8. Meijers Marcos. *Tool Support for Object-Oriented Design Patterns* – Master Thesis, INF-SCR-96-28, Computer Science Department, Utrecht University, Netherlands, August 1996.

9. Gabriela Aranda, and Richard Moore. *GoF Creational Patterns: A Formal Specification*. Technical Report 224, UNU/IIST, P.O. Box 3058, Macau, August 2000.
10. Andres Flores, Richard Moore. *Analysis and Specification of GoF Structural Patterns.* Proc. of 19th IASTED (AI2001), Innsbruck, Austria, Febrero 2001 (625-630). From Tech. Rep. 207, UNU/IIST, Macau, August 2000.
11. Luis Reynoso and Richard Moore. *GoF Behavioural Patterns: A Formal Specification*. Technical Report 201, UNU/IIST, P.O. Box 3058, Macau, May 2000.
12. Eden A. H., Hirshfeld Y., Yehudai A. *Towards a Mathematical Foundation for Design Patterns*, "http://www.math.tam.ac.il/~eden/bibliography"
13. Eden A. H., Gil J., Yehudai A. *A Formal Language for Design Patterns*. The 3rd Annual Conference on the Pattern Languages of Programs - PLoP'96 (Technical Report WUCS-97-07 University of Washington).
14. Tommi Mikkonen. *Formalizing Design Patterns*. Proceedings of the 20th International Conference on Software Engineering, IEEE Computer Society Press, pg. 115-124, 1998.
15. Harry, Andrew - *Formal Methods Fact File VDM and Z* - John Wiley & Sons, 1996.
16. K Lano, J Bicarregui, S Goldsack. *Formalising Design Patterns*. BCS Northern Formal Methods Workshop, EWIC Series, Springer Verlag, 1997.
17. DisCo project homepage. *At URL http://disco.cs.tut.fi*.
18. Eden A. H., et. al. *Precise Specification and Automatic Application of Design Patterns*. The 12th IEEE International Automated Software Engineering Conference - ASE 1997. Available at http://www.math.tau.ac.il/~eden/
19. Eden A. H., et. al. A. *LePUS - A Declarative Pattern Specification Language*. Tech.Rep. 326/98, Available at http://www.math.tau.ac.il/~eden/bibliography.html#lepus. Dep. Computer Science, Tel Aviv University, Israel.
20. A. Lauder, S. Kent. *Precise Visual Specification of Design Patterns*. Proc. of 12th ECOOP, Brussels, Belgium, LNCS 1445. Springer-Verlag of London Math. Society 2 (42), 1936-7, pp. 230-236. July 1998.
21. McC. Smith J. and Stotts D. *Elemental Design Patterns: A Formal Semantics for Composition of OO Software Architecture*. Proc. of 27th Annual IEEE/NASA Soft. Engin. Workshop. Greenbelt, MD, USA. December 2002.
22. Dan Van Hung, Chris George, Tomasz Janowski, Richard Moore. *Specification Case Studies in RAISE*. Chapter: *Object-Oriented Design Patterns*, R. Moore, A. Flores, et.al. FACIT Series, Springer-Verlag, Great Britain, 2002.
23. Booch Grady. *Object Oriented Analysis and Design With Applications*. 2nd edition. Benjamin/Cummings. 1994.
24. Booch G., J. Rumbaugh, I. Jackobson. *UML: A Unified Notation Language*, version 1.0. 1997. Available electronically from: http://www.rational.com/uml
25. Rumbaugh J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. *Object Oriented Modeling and Design*. Prentice Hall. 1991.
26. Holland Ian M. *Specifying reusable components using Contracts*. In: *ECOOP '92 Conference Proceedings*, Lecture notes in Computer Science #615, Springer-Verlag, 1992.
27. Alain Le Guennec, Gerson Sunyé, Jean-Marc Jézéquel. *Precise modeling of design patterns*. In Proceedings of UML 2000, volume 1939 of LNCS, pages 482-496. Springer Verlag, 2000.
28. UML Consortium: Object Constraint Language Specification, *http://www.rational.com* (1997)
29. M. Abadi, L. Cardelli. *An imperative object calculus*. Proc. of TAPSOFT '95, N° 915, Springer-Verlag, pgs. 471-485. 1995. Available at: http://citeseer.nj.nec.com/abadi96imperative.html
30. J. Gil, J. Howse and S. Kent. *Constraint Diagrams: A Step Beyond UML*. In Proceedings of TOOLS USA '99, IEEE Computer Society Press, 1999.
31. A. Flores, *et al. Tool Support for Verifying Applications using Object-Oriented Patterns*. WICC'02, 4to Workshop de Investigadores en Ciencias de la Computación. UNS. Bahía Blanca, Argentina. Mayo 2002.
32. A. Flores, *et al*. *Component-based Tool for Verifying Applications using Object-Oriented Patterns*. Invitado a publicarse en JCS&T'02, Journal of Computer Science and Technology 2002. Disponible en http://journal.info.unlp.edu.ar/default.html, Octubre 2002.
33. A. Flores, Pablo Fillottrani. Modelos Formales para Patrones de Diseño OO. Reporte Técnico R010, Departamento de Ciencias de la Computación, Universidad Nacional del Comahue, Neuquén, Argentina. Junio 2003.