



Departamento de Informática
Universidad de Castilla-La Mancha
Tesis Doctoral

**CARACTERIZACIÓN DEL
CONOCIMIENTO EN DISEÑO DE
MICRO ARQUITECTURAS
ORIENTADAS A OBJETOS**

Ciudad Real, Junio de 2004

Doctorando: D. Javier Garzás Parra

Director: Dr. D. Mario G. Piattini Velthuis

Capítulo 3 Propuesta de Ontología del Conocimiento en Diseño de Micro Arquitecturas OO

Una vez revisado el estado del arte del conocimiento en diseño de micro arquitecturas OO nuestro objetivo es caracterizar éste y para ello crearemos una ontología, con el propósito de proporcionar orden y unificación dentro de la confusión existente sobre los elementos que forman el conocimiento relacionado con el diseño OO.

3.1 Presentación y Objetivo

Una Ontología describe el conocimiento de un dominio de forma genérica, proporcionándole analizabilidad añadida, especifica una conceptualización, define, describe y relaciona conceptos (Gruber, 1991). Para nosotros, en el diseño de micro arquitecturas OO una ontología aportará beneficios potenciales en la:

- Comunicación (entre sistemas computacionales, humanos y ambos).
- Inferencia computacional (representación interna, algoritmos, etc.).
- Reutilización y organización del conocimiento (dominios estructurados, repositorios, etc.).
- Estructura y unificación del conocimiento.
- Enseñanza de los conceptos del diseño OO.
- Resolución de incompatibilidades terminológicas.
- Globalización del conocimiento.

Por otro lado, la ontología se construirá con el objetivo de integrarse junto con otras sobre la "gestión de proyectos de mantenimiento", "flujos de trabajo" y de la "medida" (Ruiz, 2003; García *et al.*, 2004) en un entorno global para la gestión de procesos software.

Para describir la ontología utilizaremos las fases que especifica la metodología REFSENO (Tautz y Von Wangenheim, 1998), similares al ciclo de vida software propuesto por el estándar IEEE 1074 (IEEE, 1997), estas son:

- a) Planificación (ver 3.2 "Planificación de la Construcción de la Ontología", Pág. 81).
- b) Especificación (ver 3.3 "Especificación", Pág. 83).
- c) Conceptualización (ver 3.4 "Conceptualización de las Entidades del Conocimiento en Diseño de Micro Arquitecturas OO", Págs. 86, y 3.5 "Conceptualización de las Relaciones entre los Elementos que Forman el Conocimiento", pág. 92).
- d) Implementación (ver Capítulo 7 "Automatización", Pág. 186).

3.2 Planificación de la Construcción de la Ontología

Para la construcción y descripción de la ontología nos basamos en dos metodologías: REFSENO y Helix-Spindle.

REFSENO (Tautz y Von Wangenheim, 1998) es una adaptación mejorada de Methontology (Fernández *et al.*, 1997 y Gómez-Pérez, 1998), y la hemos usado principalmente a la hora de describir la ontología.

Por otro lado, el ciclo de vida y planificación de la construcción de la ontología que hemos seguido coincide con el propuesto recientemente por el modelo de procesos de ingeniería ontológica Helix-Spindle (Kishore *et al.*, 2004), donde combinamos una aproximación teórica y pragmática para el desarrollo de la ontología. Así, el desarrollo de la ontología se ha basado en un ciclo incremental e iterativo (del estilo al ciclo de vida descrito en diversas metodologías para la construcción software).

En un primer nivel la construcción de la ontología se basa en dos fases¹¹ (ver Figura 7): Concepción y Elaboración. Durante la fase de Concepción la ontología es inicialmente conceptualizada y especificada usando una meta-ontología informal (notación), por ejemplo el lenguaje natural. En la fase de elaboración la ontología es refinada y elaborada expresándola usando una meta-ontología semi-formal, en nuestro caso UML. Por otro lado, cada una de las dos fases anteriores consta tanto de una etapa de construcción como otra de pruebas, ambas iterativas e incrementales por ciclos. Hay que destacar la iteración de la fase de pruebas, que nos ha permitido una realimentación y refinamiento constante de la ontología, con lo que los problemas se identificaron en fases tempranas.

Todo el ciclo anterior, así como la temporalidad de cada una de las fases, se muestra en la Figura 7. Los resultados intermedios de esta investigación, que nos sirvieron de realimentación sucesiva hasta elaborar la versión final de este trabajo, se muestran en el apéndice "I.1 Evolución de la Ontología " (Pág. 202).

¹¹ Helix-Spindle contempla, además de las fases de concepción y elaboración, una última fase de definición, más centrada en elaborar una representación formal tipo BNF.

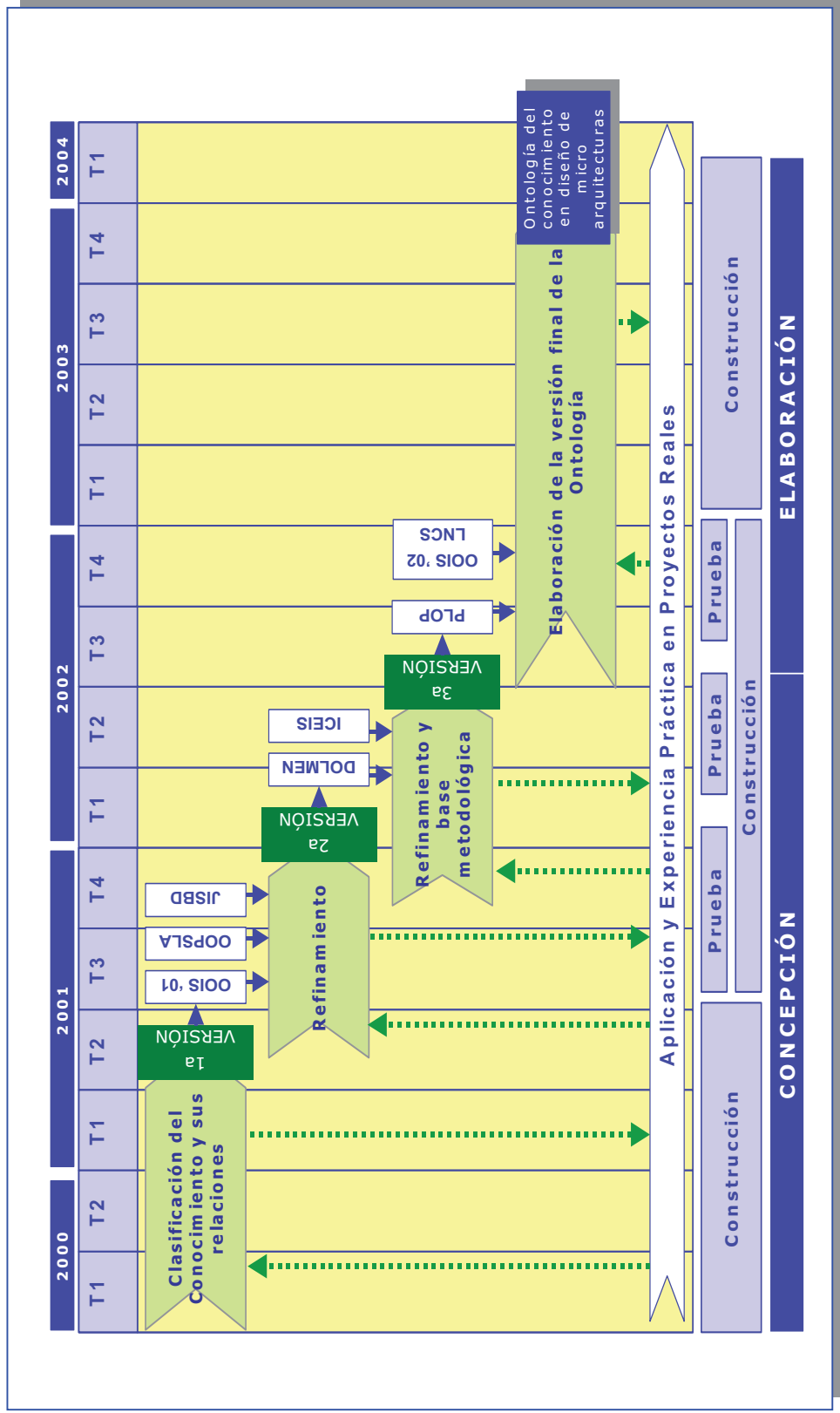


Figura 7. Ciclo de Vida de la Construcción de la Ontología.

3.3 Especificación

3.3.1 Información General

Se describe el propósito de la ontología, su ámbito, información de carácter general y fuentes de conocimiento usadas, esta información se resume en la Tabla 13.

Concepto	Valor
Dominio	Diseño de Micro Arquitecturas Orientadas a Objetos.
Autor	Javier Garzás.
Propósito	Ontología para caracterizar los elementos de conocimiento vinculados al diseño de micro arquitecturas OO.
Nivel de formalidad	Semi-formal. UML y REFSENO.
Ámbito	Lista de Conceptos: Conocimiento declarativo, Conocimiento operacional, Reglas, Patrones y Refactorizaciones.
Fuentes de Conocimiento	Riel (1996), Gamma <i>et al.</i> (1995), Fowler (2000), Martín (1995), etc.

Tabla 13. Información General sobre la Ontología del Conocimiento.

3.3.2 Tipo y Representación

Antes de mostrar la ontología propuesta debemos seleccionar su modelo y modo de representación.

Respecto al modelo de ontología, distinguimos cuatro tipos (Jurisica *et al.*, 1999):

- *Estáticas*, comprenden aspectos estáticos de una aplicación, descritos en términos tales como entidad, atributo, relación, etc.
- *Dinámicas*, comprenden aspectos dinámicos dentro de la aplicación, descritos en términos de proceso, actividad, acción, plan, recurso, etc.
- *Intencionales*, describen el mundo de los agentes (humanos o de otro tipo). Descrito en términos como cree en, quiere, prueba, argumenta, etc.
- *Sociales*, describen conjuntos sociales en términos de relaciones entre agentes, tales como autoridad, responsabilidad, actor, posición, rol, etc.

De acuerdo con lo anterior nuestra Ontología estará dentro de las denominadas estáticas.

Respecto a la representación, varios formalismos han sido desarrollados para expresar ontologías (Cranefield *et al.*, 2001), como son el *Knowledge Interchange Format* (KIF) o el *Knowledge Representation Language* descendiente del KL-ONE (Brachman y Schmolze, 1985). REFSENO utiliza unos diagramas parecidos a UML. Nosotros presentamos la ontología usando directamente la notación UML, propuesta como lenguaje de representación ontológica en ingeniería del software (Guizzard *et al.*, 2002), ya que, como afirman Cranefield *et al.* (2001), *"la gran comunidad de usuarios y el soporte comercial de los estándares OO garantizan la investigación en técnicas de modelado de objetos y estándares para el desarrollo de ontologías"*.

3.3.3 Situación y Contexto

Antes de describir nuestra Ontología es importante explicar cuál es el ámbito y alcance de la misma. Para ello, y con el objetivo de evitar ambigüedades, tomamos como marco de referencia el proyecto SWEBOK (Abran *et al.*, 2001) para ver dónde se situaría una Ontología del Conocimiento en Diseño de Micro Arquitecturas OO.

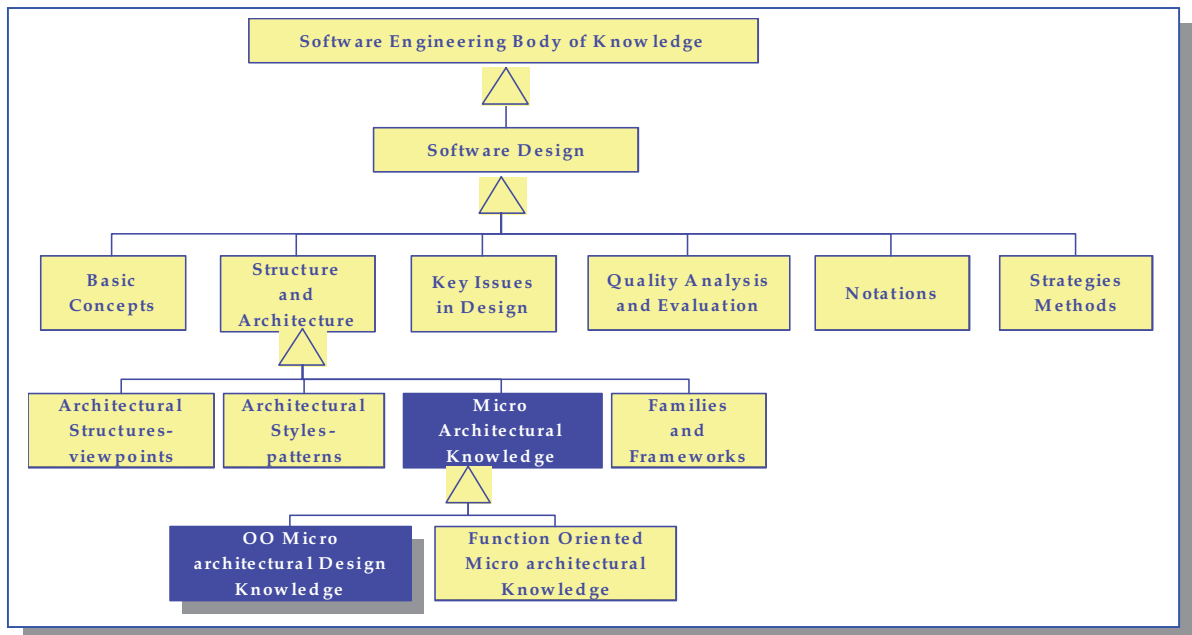


Figura 8. Subáreas del Diseño Software según el proyecto SWEBOK.

Nuestra ontología está situada bajo el área de "Software Design" (ver Figura 4, pág. 20), lo que no se ve tan claro es bajo cuál de las 6 sub-áreas de "Software Design". No obstante, estudiando el proyecto encontramos un lugar para dos elementos de conocimiento: reglas dentro del área "Basic Concepts" (bajo el nombre de "Enabling Techniques"), y patrones de diseño bajo el área de "Structure and Architecture", pero no se ve tan claro el lugar de otros conceptos como por ejemplo las refactorizaciones (sólo citado en el SWEBOK de forma breve en el área de Mantenimiento). Ya que nuestra ontología se centra en micro arquitecturas, y tras repasar las otras áreas, decidimos que el mejor lugar es "Structure and Architecture", donde se centran las estrategias de diseño arquitectónico, y proponemos modificar esta subárea para que generalice el conocimiento existente en la misma dando cabida de forma explícita a otros elementos aparte de los patrones, que son sólo uno de los núcleos que forman el "Micro Architectural Knowledge".

Así, como se ve en la Figura 8, nuestra idea es introducir un área más genérica dentro de "Structure and architecture", la de "Micro Architectural Knowledge" y dividir esta en Object Oriented y Function Oriented. Nuestra ontología estructura la parte OO. Bajo esta nueva área, "Micro Architectural Knowledge", también se situarían las reglas de diseño de micro arquitecturas, distintas de los que SWEBOK considera bajo "Basic Concepts" ("Enabling Techniques"), ya que estos son más propiedades básicas aplicables de forma general al diseño que estrategias concretas de estructura y arquitectura.

3.4 Conceptualización de las Entidades del Conocimiento en Diseño de Micro Arquitecturas OO

Antes de establecer una caracterización y una base metodológica sobre el conocimiento en Diseño OO es necesario precisar qué es cada uno de los elementos que componen este conocimiento y en qué se diferencian.

3.4.1 Conocimiento Declarativo y Operativo

Como se expuso en el Estado del Arte (Capítulo 2, Pág. 28), aparte del conocido concepto de "patrón" existen muchos otros conceptos que agrupan un importante conocimiento sobre el diseño de micro arquitecturas OO, y entre estos se encuentran las "heurísticas", "mejores prácticas", "lecciones aprendidas", "*bad smells*", "refactorizaciones", etc., los cuales no pueden obviarse si pretendemos obtener el mayor beneficio de nuestra experiencia en diseño OO.

El problema que encontramos es que salvando al concepto de patrón, el resto de elementos que podríamos asociar al conocimiento en diseño de micro arquitecturas OO están pobremente clasificados, diferenciados, accesibles, etc. Estos problemas dificultan su utilización, comprensión y ampliación. Además, muy pocos autores han tratado esta problemática, y apenas hemos podido encontrar algún estudio, salvando un párrafo de Priestley (2001), que apunta como "*el conocimiento acumulado por la comunidad OO se divide en dos categorías: los principios de alto nivel, ampliamente aceptados, y los patrones, que documentan distintos tipos de conocimiento de diseño*".

Con el objetivo de lograr una clasificación unificada del conocimiento, hemos observado como, si bien existen numerosos términos relacionados con lo que podríamos denominar conocimiento esencial acumulado de la experiencia en el diseño de micro arquitecturas, podemos hacer una primera agrupación en dos conjuntos:

- Por un lado tenemos los denominados Principios, Heurísticas, Patrones, Malos Olores (*Bad Smells*), Lecciones Aprendidas, Mejores Prácticas, Buenos Hábitos y similares, todos de carácter **declarativo**.
- Por otro, podemos encontrar otros conceptos como la refactorización. Este expresa una cantidad de conocimiento acumulado sobre cuál es la mejor manera de realizar cambios en el software, un método de realizar estos, transformación parametrizada de

un programa preservando su funcionalidad (Opdyke, 1992). Así supone una operación o proceso, es decir, conocimiento **operativo**. Hay que resaltar que nos referimos a refactorizaciones en el ámbito de diseño (*design refactoring*), no de código.

Podemos hacer una primera clasificación y dividir el conocimiento esencial basado en la experiencia en el diseño de micro arquitecturas software OO en dos bloques: el conocimiento *declarativo* y *operativo*. El primero nos dice **qué** hacer y el segundo **cómo, ambos según la experiencia**.

3.4.2 El Concepto de Regla de Diseño de Micro Arquitecturas OO

Centrándonos exclusivamente en el conocimiento declarativo y sin tomar en cuenta a los patrones, quedándonos sólo con Principios, Heurísticas, *Bad Smells*, etc., hemos observado que en esencia estos siempre tienen la misma estructura común, y aunque diferentes autores los han denominado de distinta forma y existe una amplia heterogeneidad entre las formas de describir estos componentes del conocimiento, no existe diferencia sustancial entre los mismos: todos atienden a la estructura y forma de una Regla, en la que exponen una condición para la que de cumplirse se ofrece una recomendación, destacar la "recomendación" de la Regla, que no es una solución como la del Patrón (en la sección 3.4.3.1, Pág. 88, se trata con detalle las diferencias entre patrón y regla)

Al no haber encontrado elementos bibliográficos que detallen de forma precisa heurísticas, principios y similares, y mucho menos estudios que traten a todos los núcleos de conocimiento de manera relacionada, nosotros aportamos la siguiente **definición** de "**Regla de Diseño de Micro Arquitecturas OO**", y tomamos para patrón y refactorización cualquiera de las detalladas en las secciones 2.3 y 2.4, respectivamente.

3.4.2.1 Definición de Regla de Diseño de Micro Arquitecturas OO

Regla de Diseño de Micro Arquitecturas OO

Ley basada en la experiencia que mejora la calidad de un diseño. Se caracteriza por:

- Ser un argumento declarativo perteneciente al uso del conocimiento general y práctico adquirido con la experiencia.
- Ser una ley que de violarse supone una menor calidad en el diseño, pero que no es problema concreto del mismo. La ausencia de aplicación de una regla no supone que el diseño no funcione, aunque sí puede reflejar una menor calidad. Por tanto es un elemento no intrínseco a los elementos base de la OO y siempre puede introducirse o no en un diseño.
- Ser una fuerza que fundamenta a una buena micro arquitectura de diseño.

3.4.3 Comparativas entre Elementos de Conocimiento

3.4.3.1 Diferencias entre Reglas y Patrones de Diseño

Con el objetivo de aclarar el concepto de regla es importante describir las diferencias entre estas y los patrones, ya que estos son dos de los núcleos de conocimiento más importantes y muchas veces sus diferencias ofrecen confusión.

Sólo unos pocos trabajos hacen referencia a esta diferencia, pero de forma vaga y generalmente a la hora de diferenciarlos de los patrones. Así, Appleton (2001), comenta como *"los patrones son más que heurísticas, reglas o algoritmos. Las Heurísticas y los Principios frecuentemente participan en las fuerzas y/o fundamento del patrón, pero son sólo uno de los elementos del patrón. Los patrones son el proceso que genera la solución, pero pueden generar cualquiera de un vasto número de soluciones variantes (sin repetirse dos veces la misma)"*. También Coplien (1998) expresa como *"Las reglas por lo común no están soportadas por una base de fundamento, no se introducen en un contexto. Una regla puede ser parte de la solución en la descripción de un patrón, pero la solución de una regla no es ni necesaria ni suficiente. Los patrones no son diseñados para ser ejecutados o analizados por los ordenadores, cosa que sí puede imaginarse para una regla: los patrones son aplicados por arquitectos con perspicacia, experiencia y con sentido de la estética"*.

A continuación mostramos nuestra visión de estas diferencias según los estudios que hemos realizado sobre el conocimiento:

- Un patrón de diseño es un núcleo de conocimiento más formalizado que una regla de diseño. De hecho la descripción de un patrón es siempre más rica que la de una regla, de ahí que las reglas se encuentren descritas de forma más ambigua. Pero nosotros pensamos que son aplicables a un regla, la mayoría de las veces, descripciones del tipo de las descritas para un patrón en el epígrafe 2.3.2 (página 36), por lo que hemos resuelto este problema con un catálogo de reglas unificado y detallado (Capítulo 4 pág. 108).
- Los patrones suponen un problema de diseño y la solución asociada al mismo. Las reglas de diseño son más *recomendaciones* que problemas, normas que debería cumplir un diseño y que no tienen porqué suponer algo erróneo en el mismo.
- Una regla podría asociarse más con una frase del tipo "*el diseño debería cumplir*", mientras que un patrón se asociaría con "*el diseño debe resolver*" (Pescio, 1997).

3.4.3.2 Comparativa de los Elementos de Conocimiento

A continuación se muestran ciertas comparativas entre los núcleos de conocimiento, con el fin de completar de forma más precisa las definiciones anteriormente expuestas. En la Tabla 14 se comparan los distintos núcleos de conocimiento en función de los siguientes atributos:

- Abstracción, grado en que el elemento de conocimiento es más o menos cercano a la implementación o a la codificación.
- Madurez, grado en que el elemento de conocimiento ha sido estudiado, ha sido fruto de investigaciones y se dispone de experiencia sobre el mismo.
- Sistematización, grado en que se disponen de metodologías y procedimiento sobre cómo aplicar el elemento de conocimiento.
- Popularidad, grado en que el elemento es conocido, aplicado y usado en la comunidad software.
- Grado de Automatización, grado en que el elemento de conocimiento es soportado en herramientas software que faciliten su aplicación.

	DECLARATIVO		OPERATIVO
	Regla	Patrón	Refactorización
Abstracción	+++	++	+
Madurez	+++	+++	+
Sistematización	+	++	+++
Popularidad	+	+++	++
Grado de Automatización	+	++	+++
<i>Notación: +++ Alto ; ++ Medio; + Bajo</i>			

Tabla 14. Caracterización de diferentes núcleos de conocimiento.

3.4.4 Los Atributos que Describen a las Entidades del Conocimiento

Con la idea de reutilizar las ideas existentes, tomamos como base para determinar los atributos que caracterizan a las entidades del conocimiento las secciones que el catálogo de Gamma *et al.* (1995) utiliza (muchos de los cuales también utilizan Buschamnn *et al.*, 1996) en la descripción de un patrón, estas son:

- Nombre (Name).
- Propósito (Intent).
- También Conocido Como (Also known as).
- Motivación (Motivation).
- Estructura (Structure).
- Aplicabilidad (Applicability).
- Participantes (Participants).
- Colaboraciones (Collaborations).
- Consecuencias (Consequences).

- Implementación (Implementation).
- Código de Ejemplo (Sample code).
- Usos Conocidos (Known Uses).
- Patrones Relacionados (Related Patterns).

La mayoría de los anteriores pueden generalizarse y ser elemento común a todas las entidades del conocimiento en diseño de micro arquitecturas OO, mientras que otros son sólo aplicables a una entidad concreta y con adaptaciones. Consideramos como atributos comunes a todas las entidades del conocimiento en diseño de micro arquitecturas OO a los anteriores con la excepción de los siguientes:

- **Estructura**, este atributo contiene la solución propuesta por un patrón, por lo que creamos el atributo **Recomendación** para reglas (enfaticando la recomendación de la regla frente a la solución del patrón; Pescio, 1997) y **Mecánica** para las refactorizaciones (tomamos el nombre de este atributo del catálogo de refactorizaciones de Fowler (2000)).
- **Implementación**, se sustituye por **Mecánica**, y que como se vio es aplicable a la refactorización (recordar que tratamos con refactorizaciones a nivel de diseño y no de código).
- **Código de Ejemplo**, es sustituido por **Diseño de Ejemplo**, ya que la ontología se centra en el diseño.
- **Patrones Relacionados**, que se generalizará en distintas relaciones o asociaciones entre entidades del conocimiento. Estas relaciones se estudian en el siguiente epígrafe.

Destacar también que para las refactorizaciones es importante la información sobre las entradas a la operación (argumentos) y la información sobre las precondiciones para su aplicación. Esta información viene reflejada en los atributos **Participantes** y **Aplicabilidad** respectivamente, que son comunes a todas las entidades del conocimiento.

En la Figura 16, donde se muestra la ontología de forma completa, pueden observarse los atributos del conocimiento según las anteriores consideraciones.

3.5 Conceptualización de las Relaciones entre los Elementos que Forman el Conocimiento

En esta sección nos centramos en detallar la importante relación que existe entre los elementos del conocimiento y para ello utilizaremos un ejemplo (Figura 9).

Sean dos clases A y B bajo un esquema como el detallado en la Figura 9, parte 1, en el que se detecta la violación de la **"Regla de SI hay Dependencias de Clases Concretas"**¹² (esta regla se describe con detalle en la Pág. 110, donde hemos elaborado un catálogo unificado de reglas de diseño). La no violación de esta regla sería una mejora potencial para el diseño, donde si bien el diseño continúa funcionando con la violación de esta regla, muestra una menor calidad.

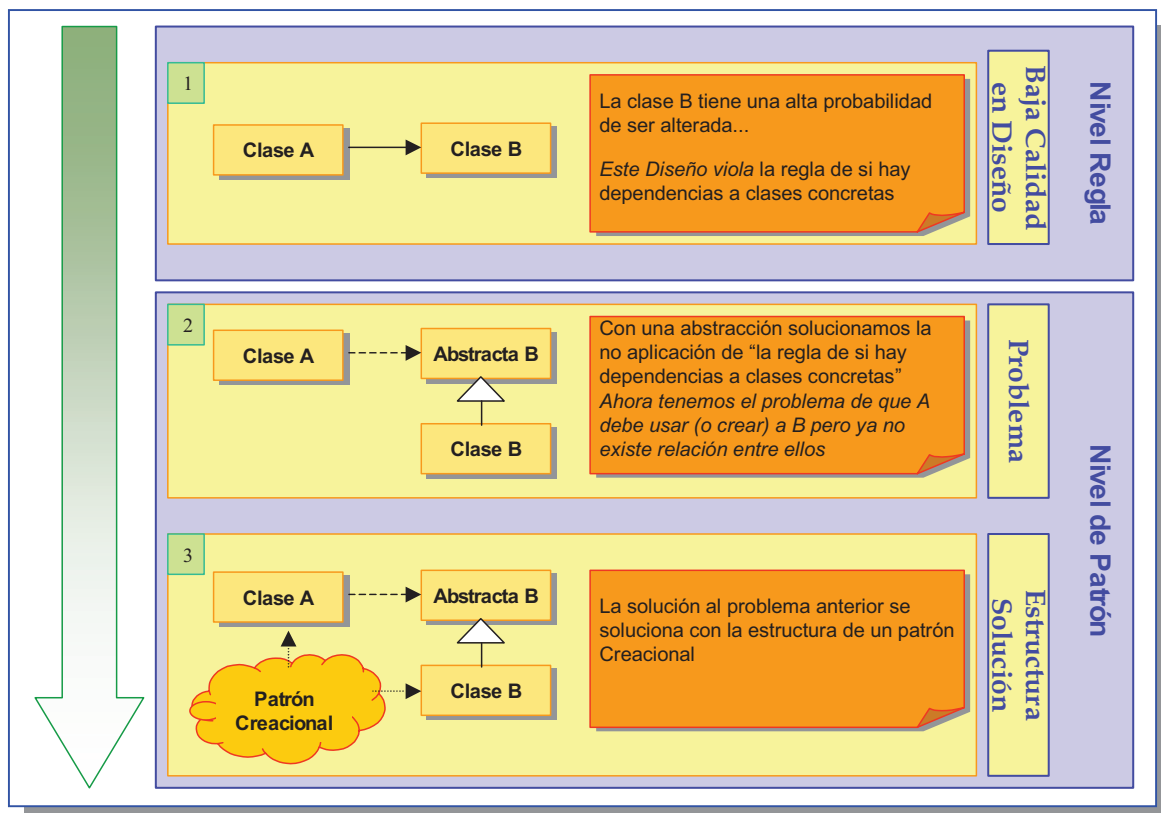


Figura 9. Ejemplo de la relación entre reglas y patrones de diseño.

¹² Esta regla también es conocida por DIP (Dependency Inversion Principle - Inversión de la Dependencia) (Martin, 1995) o por Programe una interfaz, no una implementación (Gamma *et al.*, 1995). Nosotros la hemos unificado en formato y nombre respecto a otras existentes en la bibliografía. De forma resumida la regla dice que SI hay dependencias de clases concretas ENTONCES hacer que estas dependencias sean de abstracciones. Observar también que para unificar y facilitar su aplicación nombraremos a las reglas por su parte antecedente.

3.5.1 Relaciones entre el Conocimiento Declarativo (Reglas y Patrones)

Para cumplir la regla del ejemplo, esta nos dice que hay que depender de abstracciones, parte 2 (Figura 9), una solución que lleva implícita un problema: ¿cómo mantener el uso que anteriormente hacía la clase A de la clase B si ahora no están directamente conectadas? Una vez detectado el problema originado de resolver la violación de la regla encontramos que la solución pasa por aplicar un patrón Creacional (Gamma *et al.*, 1995), y es este patrón Creacional (para crear instancias y asociar objetos en la nueva situación) el que nos va a permitir que el diseño no viole la regla. En resumen podemos afirmar que en muchas ocasiones, tras introducir una regla obtenemos un nuevo diseño el cual necesita un patrón. Con todo podemos sacar la siguiente conclusión: "Aplicar una Regla Implica Utilizar un Patrón".

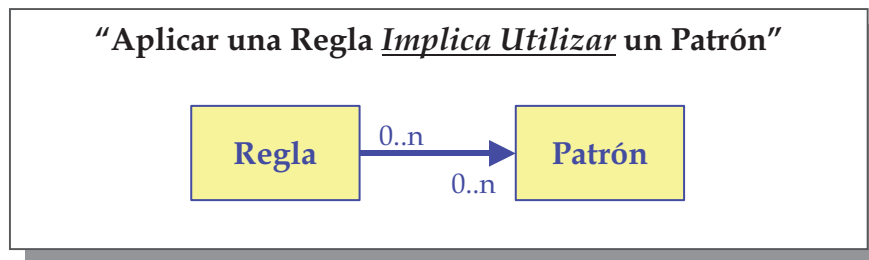


Figura 10. Relación Regla - Patrón

Hay que observar cómo esto no siempre sucede (de ahí que la cardinalidad en el destino de la implicación sea de cero a n), no todas las reglas implican introducir un patrón, un claro ejemplo de esto es cuando aplicamos reglas que trabajan sólo dentro de un módulo, como las que descomponen servicios largos, por ejemplo la "Regla de SI un Servicio Tiene Muchos Parámetros" (ver Pág. 137) (también conocida como el bad smell "Long Parameter List", Beck y Fowler (2000)). Además, no todos los patrones son implicados por reglas y un patrón puede ser implicado por varias reglas (de ahí que la cardinalidad en el origen de la implicación es de cero a n).

En este punto, visto como una regla implica a un patrón o como desde una regla podemos llegar al uso de patrones, podemos plantearnos... ¿sucede esto en el sentido contrario? En primer lugar hemos observado como muchas reglas están embebidas en un patrón, lo cual es lógico al ser las reglas "guías" de un buen diseño y al ser un patrón en sí un buen diseño... así **un patrón puede cumplir reglas** (ver Figura 11). Y un patrón puede cumplir varias reglas o incluso ninguna (lo que no significa que las viole, ya que no sería correcto en una micro arquitectura de calidad), de ahí que la cardinalidad en el destino sea de cero a n , e igualmente una regla puede ser cumplida en ninguno o varios patrones.

No obstante, esta última relación entre patrones y reglas no tiene el mismo carácter que la descrita en el sentido contrario, ya que cuando una regla es cumplida por un patrón esta

actúa como característica de la construcción del mismo y no es una relación de implicación al uso o de "cómo llegar de un patrón a una regla".

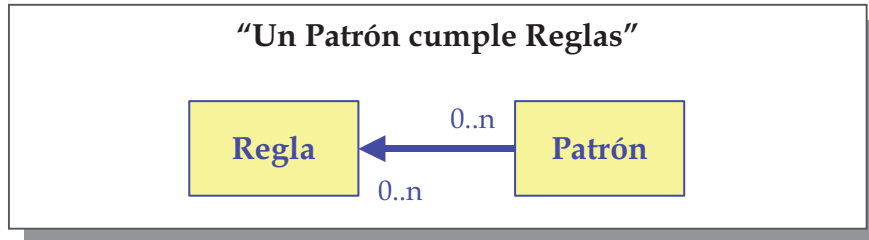


Figura 11. Relación Patrón - Regla

¿Existe entonces la relación de uso o implicación de patrones a reglas?, no, ya que si tras aplicar un patrón este nos llevase a una regla sería por alguna de las siguientes razones:

- a) Tras aplicar el patrón el diseño resultante viola alguna regla de diseño lo cual carece de sentido ya que un patrón es un diseño de calidad, que contiene reglas de mejora de calidad¹³.
- b) La regla se incumplía antes de introducir el patrón.

Con todo lo anterior en la Tabla 15 se muestra un análisis de algunas de las reglas más destacadas (la unificación, recopilación y catalogación de estas reglas es una de las aportaciones importantes de este trabajo; el catálogo se muestra en el Capítulo 4, Pág. 108) y su relación con cada patrón de los que se muestran en Gamma *et al.* (1995), donde se detalla el tipo de relación que mantienen regla y patrón. En esta tabla no sólo pretendemos observar la relación regla – patrón, sino **también observar la relación de las reglas con diseños de calidad, ya que los patrones son en si diseños de calidad.**

¹³ Al comienzo de esta investigación supusimos que esta relación (un patrón implica a una regla) existía. Pero tras un estudio más detallado nos dimos cuenta de que no era así. De hecho, la primera versión de la Tabla 15 (ver epígrafe I.1 Evolución de la Ontología Pág. 202) contenía una columna para describir esta relación.

		PATRONES																							
		Abstract F.	Adapter Class	Adapter Object	Bridge	Builder	Chain R.	Command	Composite	Decorator	Facade	Fact. Method	Flyweight	Interpreter	Iterator	Mediator	Memento	Observer	Prototype	Proxy	Singleton	State	Strategy	Tem. Method.	Visitor
Si hay dependencias de clases concretas	Implica utilizar	X			X							X							X						
	Se Cumple en	X	X	X	X	X	X	X	X	X		X	X	X	X	X		X	X	X			X	X	X
Si un Objeto tiene Diferente Comportamiento según su Estado	Implica utilizar																						X	X	
	Se Cumple en	X	X	X	X	X	X	X	X	X		X	X	X	X	X		X	X	X			X	X	X
SI una Jerarquía de Clases tiene Muchos Niveles	Implica utilizar								X	X															
	Se Cumple en	X	X	X	X	X	X	X	X	X		X	X	X	X	X		X	X	X			X	X	X
Si algo se Utiliza muy Poco o No se Utiliza	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Si una Súper Clase Conoce a Alguna de sus Subclases	Implica utilizar																						X	X	
	Se Cumple en	X	X	X	X	X	X	X	X	X		X	X	X	X	X		X	X	X			X	X	X
SI una Clase Colabora con Muchas	Implica utilizar										X														
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X
Si Un Cambio en una Interfaz Impacta en Muchos Clientes	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Si entre una Interfaz y su Implementación no hay una Abstracción	Implica utilizar																								
	Se Cumple en																								
Si una Superclase es Concreta	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X		X	X	X	X	X		X	X	X			X	X	X
Si un Servicio Tiene Muchos Parámetros	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Si una Clase es Grande	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Si Elementos de Interfaz de Usuario están en Entidades de Dominio	Implica utilizar					X	X									X		X							
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Si una Clase Utiliza más Cosas de Otra que de sí Misma	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Si una Clase Rechaza algo de lo que Hereda	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X		X	X	X	X	X		X	X	X			X	X	X
Si los Atributos de una Clase son Públicos o Protegidos	Implica utilizar																								
	Se Cumple en	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabla 15. Relaciones entre reglas y patrones.

Destacar un elemento llamativo de la anterior tabla y es que la "Regla de SI entre una Interfaz y su Implementación no hay una Abstracción" (ver Pág. 130) no implica a ningún patrón, que es lógico, pero tampoco es cumplida por algún patrón. Llama la atención porque hay patrones con jerarquías de clases, lo que induce a pensar que hay patrones que la violan. Y eso es cierto, y se debe en parte a que hemos tomado para la tabla los patrones de Gamma *et al.* (1995) los cuales están muy ligados al lenguaje C++, que no soporta el concepto de interfaz como tal (otros lenguajes como Java si lo soportan). Pensamos que, por un lado, ligar tanto un patrón de diseño a un lenguaje de implementación no es lo más positivo y, por otro, que hacer que ciertos patrones no violen esta regla es una mejora de la calidad para los mismos.

También puede observarse cómo la anterior relación de patrones ha sido ordenada alfabéticamente, pretendiendo un listado objetivo de los mismos con el que posteriormente, y sobre la base de las reglas, puedan obtenerse analogías.

En la tabla observamos que:

- Hay Reglas que implican utilizar Patrones y las cumplen estos u otros Patrones.
- Hay Reglas que no implican a Patrones y las cumplen ciertos Patrones.

De la anterior tabla pueden extraerse varias conclusiones importantes, como las siguientes:

- Se pueden clasificar los patrones según las reglas. Las reglas nos permitirían incluso lograr una catalogación de patrones diferente a la actual (limitada a su presentación alfabética en agrupaciones de baja aplicabilidad en la mayoría de los casos).
- Se pueden desarrollar hojas de chequeo de reglas que, además, valoren el diseño y ofrezcan soluciones/patrones que aseguren su cumplimiento.
- Permite guiar la utilización de patrones, ya que es más fácil saber detectar cuándo aplicar una regla que un patrón y una vez aplicada la regla es fácil llegar al patrón. Esto nos posibilita un buen uso del patrón, en su justa medida, sin abuso. Por ejemplo, el uso de la "Regla de SI hay Dependencias de Clases Concretas" (Pág. 110) implica a los patrones de creación y esto nos asegura que nuestro sistema está escrito en función de interfaces y no en función de implementaciones.
- Permite un estudio formal de micro arquitecturas, ya que se puede descomponer en fuerzas de menor granularidad cada uno de los patrones, posibilitando el estudio de elementos comunes a todos los patrones de un mismo carácter, "patrones en los patrones" o meta- patrones, las fuerzas (reglas de diseño) que conforman el patrón y cómo dependiendo de su modo de incidencia en éste el patrón puede ser de distinto carácter, ejemplos:
 - Se observa como los cinco patrones de creación, Abstract Factory, Builder, Factory Method, Prototype y Singleton (Gamma *et al.*, 1995) mantienen un núcleo de reglas idéntico, con la excepción de Singleton, el cual sólo es una estructura de implementación y se compone de una sola clase (de hecho Buschmann *et al.* (1996) lo consideran un *Idiom* o estrategia de codificación). Cómo vemos, el estudio de los principios que intervienen en un patrón nos permite, entre otras muchas cosas, una clasificación más fina y fundamentada de los patrones.
 - Se observa que cualquier micro arquitectura con alguna jerarquía de clases que se quiera considerar patrón de diseño debe cumplir (ver "se cumple en" en la Tabla 15), al menos, las siguientes reglas: "Regla de SI hay Dependencias de

Clases Concretas" (Pág. 110) y "Regla de SI una Súper-clase es Concreta" (pág. 133).

- Se observa como patrones estructuralmente idénticos como State y Strategy cumplen los mismos principios y con el mismo carácter.
- Buscar y/o validar nuevos patrones de diseño mediante el cumplimiento de ciertas reglas o meta – patrones.

Las reglas son y nos permiten extraer "buenas prácticas" OO, observando cómo se fundamentan los patrones y cómo se conectan con el diseño, pudiendo así aplicar estas "buenas prácticas" a otras situaciones del diseño, incluso evitando tener que utilizar todo el patrón para obtener alguno de sus beneficios, práctica muchas veces habitual y excesiva, que hace complejo el diseño. Por último, para fijar por completo las anteriores ideas, la Tabla 16 muestra un ejemplo de algunas reglas y los problemas que podemos encontrar a la hora de llevarlas a cabo, problemas que son resueltos por los patrones.

Reglas	Problema(s)
Regla de SI Elementos de Interfaz de Usuario están en Entidades de Dominio	Existe un acoplamiento fuerte entre emisores y receptores de eventos
	Hay que encapsular el protocolo de comunicación entre objetos
	Un objeto debe enviar información sin conocer al receptor de la misma
Regla de SI hay Dependencias de Clases Concretas	Un objeto debe crear a otro pero no existe una asociación entre ellos
Regla de SI una Súper-clase es Concreta	Un objeto debe crear a otro pero no existe una asociación entre ellos
Regla de SI un Objeto tiene Diferente Comportamiento según su Estado	Cómo permitir dinámicamente comportamiento variable
	Un algoritmo debe variar dinámicamente

Tabla 16. Ejemplo de relaciones entre Reglas y problemas resueltos por los Patrones.

3.5.2 Relaciones entre el Conocimiento Declarativo y Operativo

Volviendo al ejemplo de la Figura 9, siguiendo la regla, para mejorar la calidad del diseño se insertó en la estructura de clases una entidad abstracta, parte 2 (Figura 9)... pero ¿cómo? La manera correcta viene dada por refactorizaciones, en este caso concreto por "Extract Interface" (Fowler, 2000). De forma general observamos que hay reglas que son introducidas en el diseño, al igual que los patrones, por medio de refactorizaciones. Y con todo lo anterior, se observa de manera más clara como las refactorizaciones almacenan el conocimiento sobre cómo introducir elementos en diseños de manera controlada.

Pero las refactorizaciones también nos dicen cómo introducir de manera controlada patrones en un diseño. Esta relación se puede observar de manera más o menos implícita al leer alguno de los catálogos de refactorizaciones que se centran en el nivel de diseño (buen ejemplo de esto es parte del catálogo de Fowler (2000)). También Gamma *et al.* (1995) afirman que *"los patrones de diseño proporcionan objetivo a las refactorizaciones"*, y es que hay una relación natural entre patrones y refactorizaciones, donde los patrones pueden ser el objetivo y las refactorizaciones la manera de conseguirlo, de hecho, como afirma Fowler (2000), deberían describirse catálogos de refactorizaciones que contemplaran a todos los patrones de diseño. Así, muchas refactorizaciones concretas, tales como "Replace Type Code with State/Strategy" o "Form Template Method", se centran en introducir patrones dentro de un sistema (ver también Kerievsky, 2004; Tokuda y Batory, 2001; Tokuda, 1999).

Podemos generalizar lo anterior y afirmar que: *"El conocimiento Declarativo es introducido por conocimiento Operativo"*.

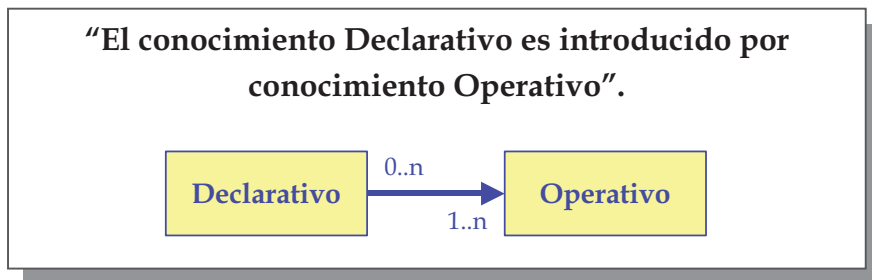


Figura 12. Relación Declarativo - Operativo

Decir que ciertos elementos del conocimiento operativo pueden no introducir conocimiento declarativo (esto sucederá generalmente con operaciones de refactorización muy simples o de granularidad fina) o pueden introducir a varios elementos del conocimiento declarativo (de ahí la cardinalidad cero a n en el origen de la relación). Y, por otro lado, todo (nótese que la cardinalidad es desde uno hasta n) conocimiento Declarativo (Reglas y Patrones) es introducido en el diseño, al menos, por un elemento Operativo (una Refactorización). Esto es bastante obvio ya que tiene poco sentido que exista un elemento de conocimiento de diseño que no pueda introducirse (la relación "Un Operativo implica a un Declarativo" no podría existir, ya que una refactorización no sabe que regla o patrón lo utilizan). Y esta es una de las razones por las que **otros conjuntos de propiedades, como los "principios" Abierto – Cerrado**

(Meyer, 1998), **de Sustitución** (Liskov y Zilles, 1974), **no podemos considerarlos Reglas sino Propiedades de un diseño**, y es que estos no se introducen en el diseño, no hay una refactorización que los introduzca, no implican a un patrón, etc., en sí son características estructurales de la OO¹⁴. Según lo anterior podemos aclarar una importante característica de las reglas (ver "Definición de Regla de Diseño de Micro Arquitecturas OO", Pág. 88): siempre se pueden introducir en un diseño, bien sea de forma directa por una refactorización o por un patrón. En síntesis, y como vimos en la Tabla 15 (Pág. 95):

- Hay Reglas que se introducen en un diseño aplicando un patrón, el cual se introduce por una refactorización. Un ejemplo es la violación de la "Regla de SI una Jerarquía de Clases tiene Muchos Niveles" (Pág. 151), la cual puede solucionarse (ver Tabla 15, Pág. 95) con patrones como Composite o Decorator (Gamma *et al.*, 1995)
- Hay Reglas que se introducen en un diseño por una refactorización, sin intervenir un patrón (ver en Tabla 15 las reglas que no implican a un patrón, Pág. 95), como son la "Regla de SI un Servicio Tiene Muchos Parámetros" (Pág. 137) o la "Regla de SI una Clase es Grande" (Pág. 140). También la "Regla de SI una Jerarquía de Clases tiene Muchos Niveles" (Pág. 151), la cual además de poder solucionarse (ver Tabla 15, Pág. 95) con patrones como Composite o Decorator (Gamma *et al.*, 1995) también, en ocasiones, puede solucionarse con refactorizaciones como "Collapse Hierarchy" (Fowler, 2000).
- Hay Reglas que se introducen por una refactorización e implican usar un patrón, como la "Regla de SI hay Dependencias de Clases Concretas" (Pág. 110).

Según lo anterior, podemos especificar más la relación entre los elementos del conocimiento operativo y declarativo. La Figura 13 muestra de manera gráfica las relaciones entre los elementos declarativos del conocimiento y los operativos (refactorización), destacando la restricción de que detrás de una Regla hay siempre, de forma directa o indirecta, una refactorización.

¹⁴ En la primera versión de la Tabla 15 (ver epígrafe I.1 Evolución de la Ontología , pág. 202), supusimos que los principios Abierto – Cerrado, sustitución, etc. eran reglas pero al completar la tabla y relacionar esta con las refactorizaciones observamos que si bien todos los patrones cumplían estos principios, estos no se introducían en el diseño, no tenían el mismo carácter las reglas.

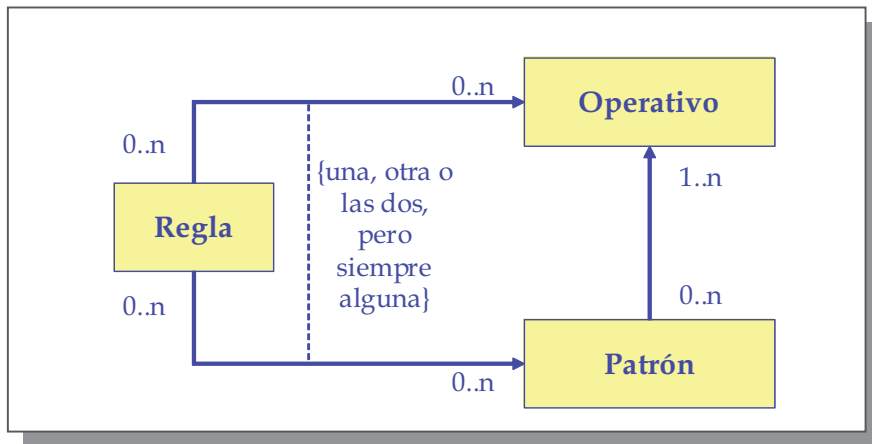


Figura 13. Detalle de la Relación Regla, Patrón y Refactorización (Operativo).

3.5.3 Relaciones Internas o Reflexivas entre los Elementos del Conocimiento

3.5.3.1 Relaciones entre Patrones

Continuando con el ejemplo de la Figura 9 y concretando un poco más una de las estrategias para cumplir la "Regla de SI hay Dependencias de Clases Concretas" podría ser usar el patrón *Abstract Factory*. Otros patrones como *Prototype*, *Factory Method*, etc., sirven más bien para realizar el *Abstract Factory* que para cumplir una regla directamente. De igual forma, nuestra regla también está relacionada con el patrón *Bridge*, de forma directa ya que este patrón soluciona problemas específicos de la "**Regla de SI hay Dependencias de Clases Concretas**". Por tanto, podemos concluir que existen patrones que directamente permiten cumplir una regla, mientras que otros patrones están más relacionados a su vez, con patrones que con reglas. Por tanto: "Aplicar un Patrón Implica Utilizar otro Patrón".

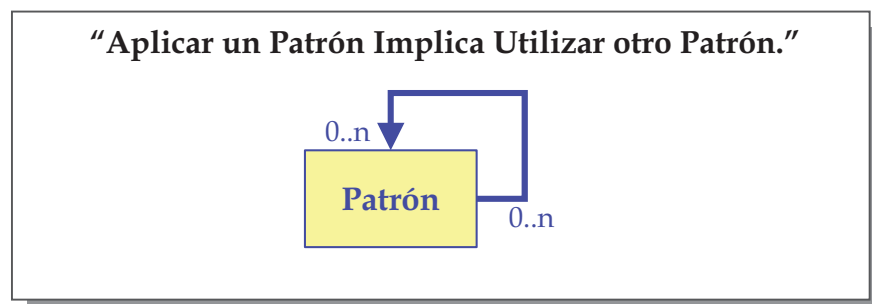


Figura 14. Relación Patrón - Patrón

Esta última relación es más conocida y estudiada, ya que está desde hace tiempo recogida en catálogos y lenguajes de Patrones, valga como ejemplo el formalismo propuesto por Amaro *et al.* (2001), Tahvildari y Kontogiannis (2002a) o el mapa de relaciones entre patrones presentado en Gamma *et al.* (1995), donde, en la contraportada, en "Patterns Map" (mapa de patrones), podemos ver las relaciones entre los patrones descritos en el catálogo, por ejemplo: de Composite a Builder, Iterator, Decorator, Flyweight y Visitor. Destacar como en este caso la cardinalidad en el destino es de cero a n, ya que, continuando con el mismo ejemplo, vemos en Gamma *et al.* (1995) que patrones como Adapter, Proxy y Bridge quedan aislados; la cardinalidad en el origen es también de cero a n, ya que hay patrones que o no son implicados o son implicados por otros varios.

3.5.3.2 Relaciones entre Conocimiento Operativo (Refactorizaciones)

Hay que destacar por un lado que esta relación es de Agregación y no de Composición, es decir, que los objetos componente tienen entidad por sí mismos, y por otro, que la cardinalidad de la relación es de 0 a n, es decir, que no todas las Refactorizaciones tienen que estar descompuestas en menores (ver Figura 15).

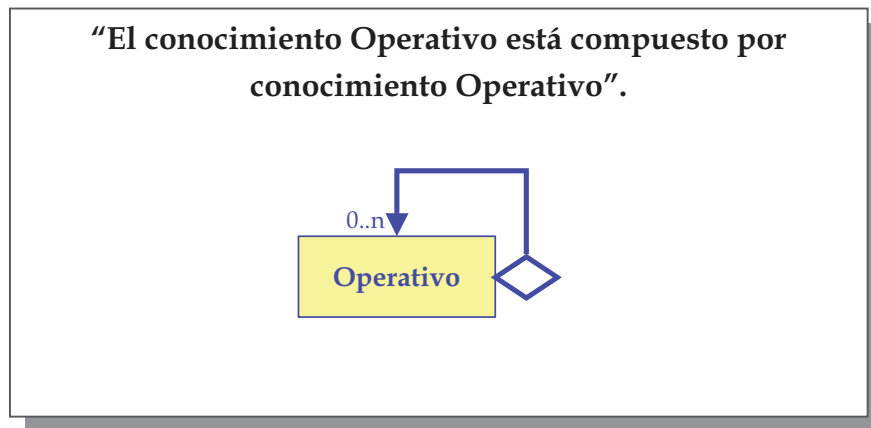


Figura 15. Relación Operativo - Operativo

3.5.3.3 Relaciones entre Reglas

Otra posible relación a valorar es la de que "una Regla implica utilizar o cumple una Regla". Hemos estudiado esta relación concluyendo en que no es posible su existencia por las siguientes razones:

- Al tener una granularidad de micro arquitectura mínima, una regla no cumple o tiene embebida a otra. Las reglas son más recomendaciones, y apenas describen una micro arquitectura como solución (ver 3.4.2, pág.87).

- Si una regla implica a otra, sería porque el resultado de aplicarla deja el diseño de forma que viole otra regla.

3.5.4 Conclusiones

Los anteriores epígrafes mostraron la relación existente entre reglas, patrones y refactorizaciones. Se ha podido observar respecto de estas relaciones cómo:

- Los patrones de diseño nos ayudan a que las reglas de diseño no sean violadas en un diseño.
- Uno de los grandes problemas a la hora de aplicar patrones es detectar los problemas que tiene el diseño, para saber dónde insertar un patrón. Las reglas, por su parte, nos detectan posibles mejoras en el diseño.
- Reglas y Patrones, de forma directa o indirecta, están soportados por refactorizaciones de diseño. Y esto es una de las características que distingue a las Reglas de otros conceptos que son más propiedades de la OO que reglas (como los principios Abierto – Cerrado o de Sustitución).

Estas relaciones forman una de las bases para establecer una ontología y una metodología de mejora de la calidad de una micro arquitectura basándose en el conocimiento.

3.6 Ontología y Glosario del Conocimiento en Diseño de Micro Arquitecturas OO

Una vez completada la exposición sobre las entidades y relaciones que forman el conocimiento en diseño de micro arquitecturas OO, la Figura 16 muestra la ontología final, usando notación UML. También puede observarse como las entidades del conocimiento se han marcado como concretas o abstractas¹⁵, según la notación UML.

Además de la descripción gráfica en UML, siguiendo la metodología REFSENO, se deben crear una serie de tablas, como la Tabla Glosario, Tabla 17, que contiene una descripción de los elementos de conocimiento que se muestran en la Figura 16. Cada fila de la tabla corresponde a un concepto. La columna etiquetada como "súper concepto" indica el concepto padre (según una relación de herencia). El concepto raíz es denotado por REFSENO mediante la palabra "Concepto". La columna etiquetada como "propósito" describe para qué es usado el concepto.

¹⁵ Destacar como la descripción del conocimiento en diseño de micro arquitecturas OO hace uso de, principalmente, dos super-entidades abstractas: declarativo y operativo. Esto permite que, si fuese necesario, pudieran incorporarse a la ontología otras entidades concretas de conocimiento. La ontología incorpora las entidades concretas regla, patrón y refactorización, ya que sobre estas existe una disciplina lo suficientemente madura para considerarlas entidades concretas. Así, por ejemplo, en el caso del conocimiento operativo pudiera suceder que alguna otra disciplina aparte de la refactorización emergiera y madurase con el tiempo, ocupando su lugar bajo la entidad operativo y junto a la entidad refactorización.

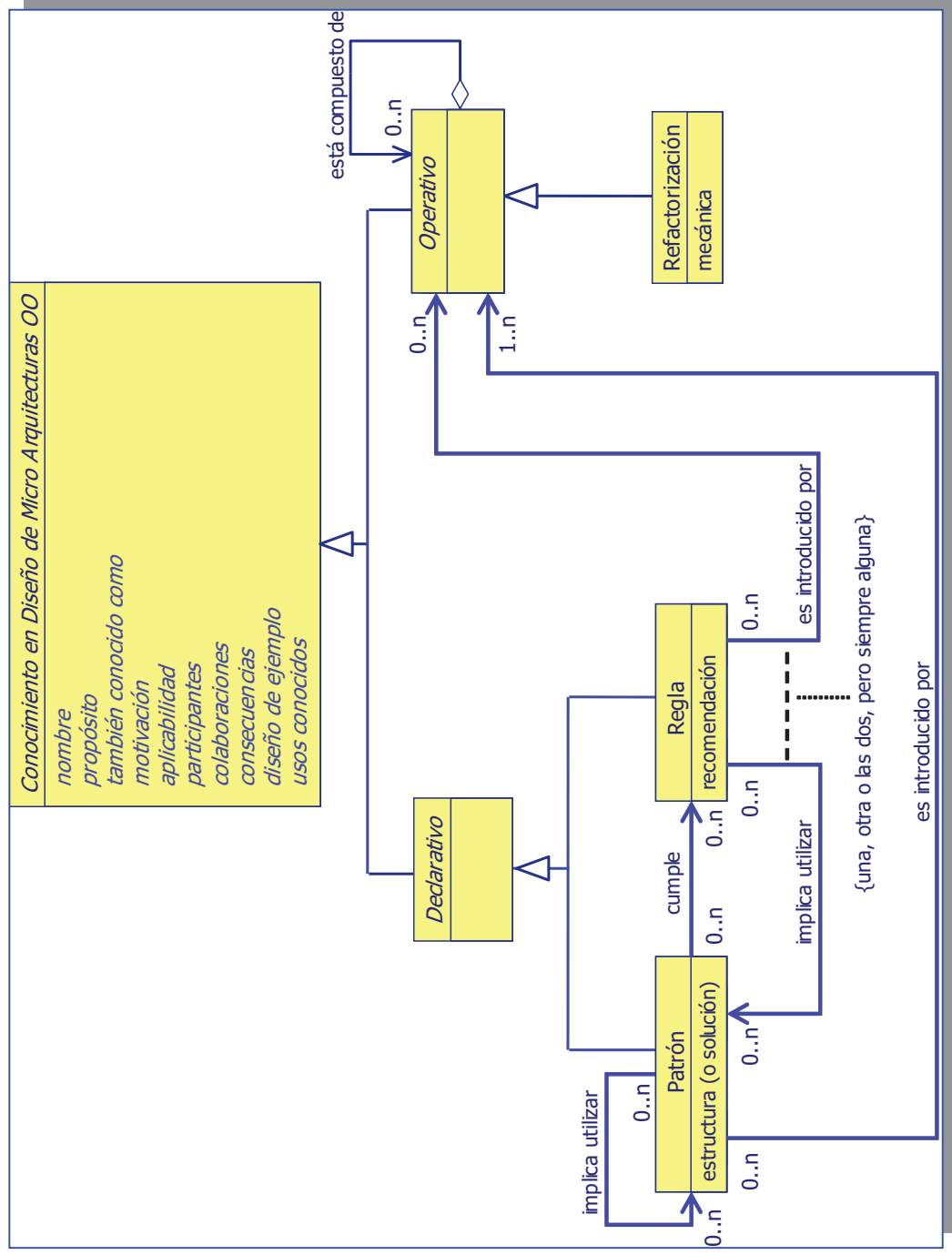


Figura 16. Ontología del Conocimiento en Diseño de Micro Arquitecturas OO.

Concepto	Súper-Concepto	Descripción	Propósito
Conocimiento en diseño de micro Arquitecturas OO	-	Conocimiento existente en el diseño de micro arquitecturas OO	Mejorar la calidad de una micro arquitectura
Declarativo	Conocimiento	Indicaciones que permiten mejorar la calidad del diseño de una micro arquitectura OO.	Dar una solución estructural para obtener un sistema OO de mayor calidad.
Operacional	Conocimiento	Conjunto de conocimiento que indica cómo realizar operaciones de transformación.	Indicar cómo realizar transformaciones.
Patrón	Declarativo	Indicación que expresa la solución a un problema en cierto contexto.	Ofrecer una solución a un problema típico en un contexto.
Regla	Declarativo	Indicación sobre buenos hábitos de diseño.	Reutilizar recomendaciones y experiencia acumulada en la mejora de la calidad de una micro arquitectura de diseño.
Refactorización	Operativo	Transformación de un diseño, preservando su funcionalidad.	Modificar un producto software para hacerlo más fácil de entender y más barato de modificar, sin cambiar su comportamiento observable.

Tabla 17. Glosario de Conceptos de la Ontología.

Nombre	Conceptos	Descripción
Es Introducido por	<input type="checkbox"/> Declarativo <input type="checkbox"/> Operativo	Un elemento de conocimiento Declarativo es introducido por uno o más elementos de conocimiento Operativo
Implica el Uso de	<input type="checkbox"/> Regla <input type="checkbox"/> Patrón	Una Regla de conocimiento implica el uso de ningún o de varios Patrones
Cumple	<input type="checkbox"/> Patrón <input type="checkbox"/> Regla	Un Patrón cumple ninguna o varias Reglas
Implica el Uso de	<input type="checkbox"/> Patrón	Un Patrón implica el uso de ninguno o varios Patrones
Está Compuesto de	<input type="checkbox"/> Operativo	El conocimiento Operativo está compuesto de ninguno o de varios conocimientos Operativos

Tabla 18. Tabla de Relaciones de la Ontología.

Concepto	Atributo	Descripción	Tipo	Card
Conocimiento en Diseño de Micro Arquitecturas OO	Nombre	Nombre del elemento de conocimiento en lenguaje natural	Cadena Texto	1
	Propósito	Objetivo del elemento de conocimiento en lenguaje natural	Cadena Texto	1
	También Conocido Como	Otros nombres del elemento de conocimiento en lenguaje natural	Cadena Texto	N
	Motivación	Justificación del elemento de conocimiento en lenguaje natural	Cadena Texto	1
	Aplicabilidad	Descripción en lenguaje natural de las situaciones en que se puede aplicar el elemento de conocimiento. También puede recoger las precondiciones para la aplicación del elemento de conocimiento.	Cadena Texto	N
	Participantes	Descripción en lenguaje natural de los elementos que participan y sus responsabilidades	Cadena Texto	1
	Colaboraciones	Descripción en lenguaje natural de cómo los participantes colaboran	Cadena Texto	1
	Consecuencias	Descripción en lenguaje natural de las consecuencias de aplicar el elemento de conocimiento	Cadena Texto	N
	Diseño de Ejemplo	Diseño que muestra un ejemplo de cómo trabaja el elemento Operativo	Cadena Texto	1
	Usos Conocidos	Descripción en lenguaje natural de situaciones donde el elemento de conocimiento se ha aplicado	Cadena Texto	1
Reglas	Recomendación	Solución propuesta por la Regla	Cadena Texto	1
Patrones	Estructura	Solución propuesta por el Patrón	Cadena Texto	1
Refactorizaciones	Mecánica	Solución propuesta por la Refactorización	Cadena Texto	1

Tabla 19. Tabla de los Atributos de la Ontología.