Modelado Estructural de Patrones de Diseño: Diagramas REP

José Luis Isla Montes[1], Francisco Luis Gutiérrez Vela[2]

[1]Departamento de Lenguajes y Sistemas Informáticos – Universidad de Cádiz
E.S. Ingeniería – c/ Chile, 1 – C.P. 11002 – Cádiz – España

[2]Departamento de Lenguajes y Sistemas Informáticos – Universidad de Granada
E.T.S. Ingeniería Informática – Periodista Daniel Saucedo Aranda s/n – C.P. 18071 – Granada – España

joseluis.isla@uca.es, fgutierr@ugr.es

Abstract.. *Software modelling languages should possess a notation, semantics and treatment adapted to design patterns, but its correct specification is a challenge for many investigators. UML treats them as parameterised collaborations, however, this approach is not exempt of problems. The goal of this work is the elaboration of a simple and intuitive model for the structural specification of design patterns and its integration in UML. The patterns specified with this model expect to be true reusable templates that can be applied in different contexts. For that, we use a visual, complete, simple and easy to learn notation. We believe that this model could be used successfully for the construction of a future tool that facilitates the definition, application, visualization and validation of patterns.*

Resumen. *Los lenguajes de modelado de software deberían poseer una notación, semántica y tratamiento adecuados para los patrones de diseño, pero su especificación correcta es un reto para muchos investigadores. UML los trata como colaboraciones parametrizadas, sin embargo, dicha aproximación no está exenta de problemas. El objetivo de este trabajo es la elaboración de un modelo sencillo e intuitivo para la especificación estructural de patrones de diseño y su integración en UML. Los patrones especificados con este modelo pretenden ser verdaderas plantillas reutilizables en diferentes contextos. Para ello, usamos una notación visual, completa, simple y fácil de aprender. Creemos que este modelo podría ser utilizado con éxito para la construcción de una futura herramienta que facilite la definición, aplicación, visualización y validación de patrones.*

1. Introducción

Los patrones de diseño han adquirido gran popularidad entre investigadores y diseñadores de software orientado a objetos. La razón principal de su éxito radica en que éstos forman parte de un vocabulario común de "buenas soluciones" perfectamente identificadas y aplicables a distintos problemas típicos de diseño que pueden encontrarse en diferentes contextos. Los patrones de diseño facilitan la reutilización del conocimiento experto como "componentes" de diseño y mejoran así la documentación, comprensión y comunicación del diseño final.

Por consiguiente, parece razonable pensar que los lenguajes de modelado deban contemplar los patrones de diseño como elementos de modelado de primer orden, estableciendo para éstos una notación, semántica y tratamiento adecuados. Paralelamente, las herramientas de diseño de software basadas en estos lenguajes deberían facilitar su definición, aplicación y validación en contextos específicos.

En el siguiente apartado se comentarán brevemente varios trabajos de importancia que han tratado el asunto de la especificación de patrones de diseño y se justificará el interés de la nueva aproximación. En el tercer apartado se pondrán de manifiesto algunos de los inconvenientes que presenta UML para representar adecuadamente la esencia de un patrón. A continuación, en el cuarto apartado, se intenta resumir la filosofía del nuevo modelo definiendo los conceptos principales que intervienen y cómo se relacionan. En el apartado quinto, como ejemplo, se muestra la aplicación de la nueva aproximación en la especificación del patrón Abstract Factory (Gamma et al., 1995). En el último apartado se plasman las conclusiones obtenidas, así como las cuestiones que quedan todavía pendientes.

2. Trabajos sobre especificación de patrones de diseño

La descripción que Gamma et al. (1995) hace de cada uno de sus patrones es buena, sin embargo, es principalmente narrativa y está basada en ejemplos concretos expresados con OMT, lo que supone un obstáculo para su tratamiento con una herramienta.

Es necesario encontrar una notación que tenga la capacidad de especificar de forma precisa la esencia/invariante de un patrón, esto es, el conjunto de restricciones estructurales y de comportamiento que lo caracterizan y que tendrán que satisfacer sus instancias.

Existen varios interesantes trabajos en esta línea, entre éstos cabe destacar:

- El lenguaje formal "LePUS" (**L**anguag**e** of **P**atterns **U**niform **S**pecification) (Eden, 2000, 2002; Eden et al., 1997, 1998) el cual se abstrae de los elementos ligados a la implementación (objetos, atributos, métodos, etc.) utilizando expresiones lógicas y una notación visual semánticamente equivalentes a partir de un universo de discurso simple donde las clases y las funciones son entidades atómicas. LePUS permite definir relaciones entre conjuntos de estas entidades (herencia, referencia, etc.), las cuales deben existir en un programa o modelo para poder conformar con la especificación realizada.
- Según Lauder y Kent (1998), la especificación de un patrón debe ser dividida en tres niveles de modelado (roles, tipos y clases) ordenados de mayor a menor nivel de abstracción. El primero captura la esencia del patrón obviando detalles específicos del dominio de la aplicación, el segundo refina el anterior añadiendo normalmente restricciones específicas del dominio y el último representa una implementación concreta. Se emplea una notación visual denominada “Diagramas de Restricción” (Kent, 1997; Lauder y Kent, 1998) para el modelado preciso de la estructura estática y dinámica de los patrones. Éstos son utilizados en combinación con otra notación gráfica de modelado, en este caso UML.

- El "Modelo de Fragmentos" de Meijers (1996) en el cual un patrón es dividido en fragmentos enlazados por roles. Un fragmento sería cada uno de los componentes relevantes de un patrón y un rol, propiedad de un fragmento, conectaría con los fragmentos que son actores de ese rol.
- Los "Diagramas de Rol" de Riehle (1996) donde los patrones son representados como colaboraciones entre roles (vista de la responsabilidades que deberá poseer un objeto) para poder abstraerse de los diagramas de clases demasiado ligados a la implementación.
- Por último, los interesantes trabajos de Le Guennec et al. (2000) y Sunyé et al. (2000) centrados en UML y donde se propone la modificación del metamodelo para usar ocurrencias estereotipadas de patrones con restricciones OCL en el meta-nivel y utilizando la base del estereotipo como contexto (la metaclase *PatternOccurrence*). Sus ideas son transferidas a la herramienta UMLAUT.

En este trabajo presentamos un modelo alternativo de especificación de patrones, basado en lo que denominamos diagramas REP. A diferencia de las demás aproximaciones, con este modelo es posible realizar una especificación precisa y visual de la estructura de un patrón mediante una sencilla extensión a UML, sin necesidad de recurrir a una notación formal como puede ser OCL. El resultado es una notación cercana al diseñador, simple, completa y muy fácil de aprender, y un modelo que trata los patrones de diseño como si fueran "verdaderas" plantillas. Por todo ello, creemos que la construcción de una futura herramienta basada en este modelo sería de un gran interés.

3. El caso de UML

Como ya es sabido, UML (Object Management Group, 2002) trata los patrones de diseño como colaboraciones parametrizadas, sin embargo, este tratamiento presenta algunas limitaciones para expresar correctamente la semántica asociada a un patrón, principalmente debido a que inicialmente éstos tienen propósitos diferentes. Así pues, si una colaboración pretende modelizar simplemente la interacción que puede existir entre un conjunto de instancias para lograr una tarea específica, un patrón tiene un propósito más general, ya que debe especificar cómo es una solución de diseño desde un punto de vista de su estructura y comportamiento. De hecho, la instanciación de una colaboración parametrizada da lugar a otra colaboración, pero la instanciación de un patrón de diseño da lugar a una porción de diseño que debe satisfacer las restricciones del patrón y que resuelve un problema concreto, pudiendo ser puramente estructural y no incluir interacción alguna.

Un patrón debería verse como una plantilla capaz de guiar al diseñador en la flexible selección, creación y validación de los elementos (clases, objetos, relaciones, atributos, métodos, etc.,) que participan en la solución de un problema típico de diseño, sin embargo, los "*ClassifierRole*" o "*AssociationRole*" de una colaboración parametrizada dependen de clasificadores o asociaciones *base* que deben existir previamente y donde los "*ClassiferRole*" o "*AssociationRole*" son una vista de éstos respectivamente. Además, sólo se pueden ligar clasificadores o asociaciones a un

parámetro con la única restricción de que el participante sea del mismo tipo (o un descendiente del tipo) que el parámetro correspondiente. Por otra parte es imposible especificar patrones con un número variable de participantes, el tipo más frecuente, debido a que el número de argumentos debe igualar el número de parámetros, etc.

Estos y otros inconvenientes son un obstáculo para poder especificar la esencia de un patrón adecuadamente. Por esta razón se propone la siguiente aproximación.

4. Propuesta de modelado y tratamiento

En este trabajo se propone un modelo diferente (Isla, 2002) para la especificación estructural de patrones de diseño y su integración con UML. Para ello, se aprovecha la capacidad expresiva de una notación muy conocida como es UML y se añade un conjunto mínimo de elementos de modelado que permiten definir el invariante de un patrón estableciendo los mecanismos apropiados para chequearlos.

Esta propuesta surge de la idea de que un patrón debería verse y usarse como una plantilla o abstracción que aplicada en un determinado contexto efectivamente ayude al diseñador en la asignación flexible, creación y validación de los distintos elementos de modelado que participan en la solución.

En la figura 1 se muestra un esquema básico con los conceptos fundamentales que han sido definidos.

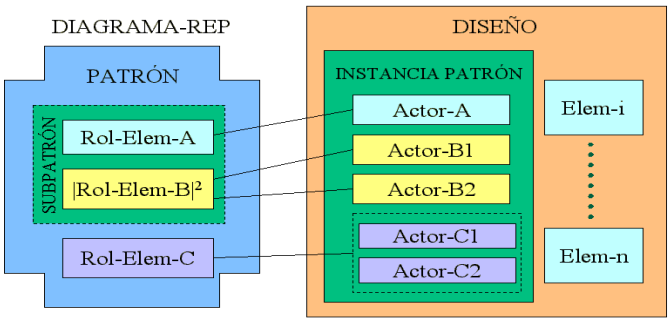

Figura 1. Conceptos básicos

Un *rol-elemento* es la especificación de un elemento de diseño que va a desempeñar un papel fundamental en la solución propuesta por un patrón. Un rol-elemento define el tipo (clase, objeto, atributo, operación, parámetro, relación, etc.) y las restricciones (estructura y/o comportamiento y/o relaciones) que deberá satisfacer dicho elemento de diseño.

Los roles-elemento se van a utilizar como piezas clave para modelar las restricciones que un patrón requiere, se podría decir que éstos van a ser sus componentes esenciales. Un rol-elemento se va a simbolizar igual que el elemento de diseño que está representando y junto a otros formará el diagrama que especificará un patrón determinado, el cual va a ser denominado *diagrama REP* (**R**oles-**E**lemento de un **P**atrón). La estructura, comportamiento y/o relaciones que presente cada rol-elemento en

dicho diagrama serán las características que deberá poseer un elemento de diseño para que pueda desempeñar ese mismo papel en una *instancia-patrón* concreta. Además, para completar el invariante de un patrón, es necesario especificar otros tipos de restricciones que deberán cumplirse por grupos de participantes (número de elementos, organización de éstos, etc.) y que podrán ser expresadas en el diagrama usando una notación especial.

Estableciendo un mecanismo que permita ligar elementos de diseño en un contexto particular con roles-elemento de un diagrama REP determinado y chequeando para todo elemento el cumplimiento de las restricciones exigidas por su asociado, se podría comprobar si ese conjunto de elementos de diseño forma una instancia-patrón del patrón en cuestión.

Cuando un elemento de diseño cumple todas las restricciones que exige un rol-elemento determinado se dice que puede ser *actor* de éste. De alguna forma se podría pensar que está asumiendo o jugando el papel de ese rol-elemento.

Como se ha indicado anteriormente, el símbolo asociado a cada rol-elemento va a ser el mismo que el del actor o actores que deberán jugarlo, sin embargo habrá que diferenciar su semántica. Los símbolos de los actores van a representar elementos que formarán parte de la implementación y, por consiguiente, sus instancias pertenecerán al sistema en ejecución, sin embargo, los roles-elemento representan elementos de diseño y serán sus actores los que sí podrán ser instanciados.

Todo rol-elemento tiene una identidad, es decir, debe poseer un identificador único dentro de su diagrama REP.

Dos roles-elemento son distintos cuando las restricciones que representan son distintas y deberán tener distinto identificador.

Dos identificadores serán distintos si presentan distinto literal o distinto formato (cursiva, subrayado, etc.)

Cada tipo de elemento de diseño se caracterizará por unas propiedades específicas, y cada tipo de rol-elemento (*rol-clase*, *rol-atributo*, *rol-relación*, *rol-operación*, *rol-parámetro*, etc.) definirá las restricciones sobre su posible actor. Así por ejemplo, un rol-clase definirá los roles-relación que tiene con el resto de roles-clase, indicará si su actor deberá ser abstracto, los roles-atributo y roles-operación que posee, etc.

Desde el punto de vista de la indispensabilidad para una instancia-patrón, un rol-elemento puede ser:

- Obligatorio. El rol-elemento debe tener algún actor.
- Opcional. El rol-elemento puede tener algún actor.

Un rol-elemento puede llevar asociado un *cardinal* que indicará la cantidad de actores que deben jugarlo.

El cardinal podrá ser:

- Fijo. Aparecerá un valor concreto.
- Variable. Aparecerá alguna variable. El valor del cardinal no se conoce a priori. Puede ser cualquier valor o puede que su valor dependa del valor que

tenga otra variable cardinal. Para que haya dependencia de valores deberán compartir alguna de sus variables. Así por ejemplo, si el cardinal de un rol-elemento es j y el de otro también es j los valores de ambos cardinales deberán siempre coincidir, si el cardinal es j en un sitio y 2*j en otro el cardinal de éste último deberá ser el doble del anterior.

Si no se especifica otra cosa, el cardinal por defecto es fijo y es igual a uno.

Una *ligadura* es la asignación de un actor a un rol-elemento.

A menudo una instancia-patrón puede presentar elementos de diseño que no se corresponden exactamente con alguno de los roles-elemento especificados en el diagrama REP del patrón, disfrazando la forma ideal de éste. A veces esto es debido a que las restricciones de un rol-elemento son satisfechas por varios elementos en conjunto. Por ejemplo un rol-relación podría ser desempeñado por varias relaciones en conjunto las cuales transitivamente satisfacen las restricciones impuestas por éste, las responsabilidades de una clase podrían estar distribuidas entre varias, el comportamiento requerido por un rol-operación puede estar repartido entre varios métodos, etc. Partiendo de esta situación, se va a llamar *ligadura simple* a la que está establecida entre un rol-elemento y un solo actor que está obligado a satisfacer todas las restricciones exigidas y *ligadura compartida* cuando la ligadura es entre un rol-elemento y varios actores que en conjunto están obligados a satisfacer todas las restricciones requeridas.

Además existen una serie de *reglas de ligadura* que deberán cumplirse:

- Un mismo actor podrá ligarse a varios roles-elementos distintos.
- Un mismo actor no podrá ligarse dos veces a un mismo rol-elemento.
- Varios actores distintos podrán ligarse a un mismo rol-elemento cuando el cardinal de éste sea superior a la unidad o cuando las ligaduras sean compartidas.
- Un mismo rol-elemento puede aparecer varias veces dentro de un diagrama REP. En tal caso, la ligadura de un actor con una de sus ocurrencias será extensiva al resto.
- Los actores correspondientes a ligaduras con roles-elementos que forman parte de otro deberán formar parte del actor ligado a ese rol-elemento (contenedor).

Se dice que una *ligadura* es *válida* cuando cumple las reglas de ligadura y el actor (ligadura simple) o actores (ligadura compartida) efectivamente cumplen los requisitos necesarios para poder desempeñar el rol-elemento designado. En otro caso será *inválida*.

Se dice que se ha realizado una *ligadura completa* sobre un diagrama REP cuando al menos todos sus roles-elemento obligatorios forman parte de alguna ligadura válida y se satisfacen sus cardinales. En caso contrario será una *ligadura parcial*.

Una *instancia-patrón* será por tanto un diagrama formado por los elementos de modelado que son actores de los roles-elemento de un diagrama REP determinado después de una ligadura completa.

Un *subpatrón* es un subconjunto definido de roles-elemento agrupados que pertenecen a un mismo diagrama REP.

Una *instancia-subpatrón* es un diagrama formado por los elementos de modelado que son actores de un subpatrón determinado después de una ligadura completa.

Un subpatrón puede llevar asociado un cardinal fijo o variable. Dicho cardinal indicará cuántas instancias-subpatrón de éste deberán existir en una determinada instancia-patrón. Cuando un subpatrón con un determinado cardinal contiene otros cardinales pertenecientes a subpatrones o roles-elemento internos a éste se dice que los *cardinales* están *anidados*.

Ha sido definida una sencilla notación visual que extiende UML para poder expresar fácilmente todas las nociones citadas anteriormente. Como ejemplo, se presenta a continuación el patrón de diseño Abstract Factory especificado según la notación empleada en Gamma et al. (1995) y se compara con su diagrama REP correspondiente según la nueva aproximación.

5. El patrón de diseño Abstract Factory como ejemplo

Este patrón (figura 2) provee una interface para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.

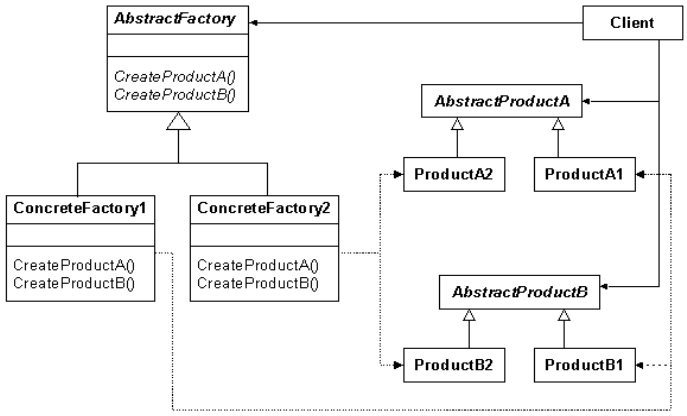

Figura 2. Patrón Abstract Factory (Gamma et al., 1995).

Esta especificación no es general porque está basada en un ejemplo concreto. En este caso, se puede observar que un número variable de clases o métodos se especifica por medio de un número fijo de éstos, en este caso dos. Esta notación no permite expresar la generalidad de un patrón sin ambigüedades.

A continuación en la figura 3, se puede observar que el diagrama REP asociado a este patrón es más general y simple. En él aparecen los roles-elemento que lo caracterizan, los cuales indican el tipo y las restricciones estructurales que deberán cumplir los actores que serán ligados.

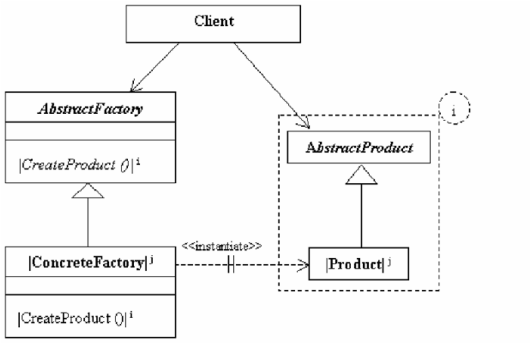

Figura 3. Diagrama REP correspondiente al patrón Abstract Factory.

Se ha aplicado una restricción sobre el número de actores que podrán jugar algunos de los roles-elemento usando un cardinal variable. En este caso, se indica que los roles-elemento "*CreateProduct()*" abstracto, "CreateProduct()" concreto y el subpatrón puede tener un número variable de actores, pero éste deberá ser el mismo para todos ya que comparten la misma variable "i". Esta situación también se presenta en "ConcreteFactory" y "Product", los cuales comparten la variable "j".

También es posible observar una restricción sobre el rol-relación que existe entre "ConcreteFactory" y "Product". Esta restricción es representada a través de dos líneas paralelas que cortan perpendicularmente la línea del rol-relación e indica que éste es biyectivo, es decir, las relaciones entre los conjuntos de actores del rol-elemento origen y destino deben formar una aplicación biyectiva.

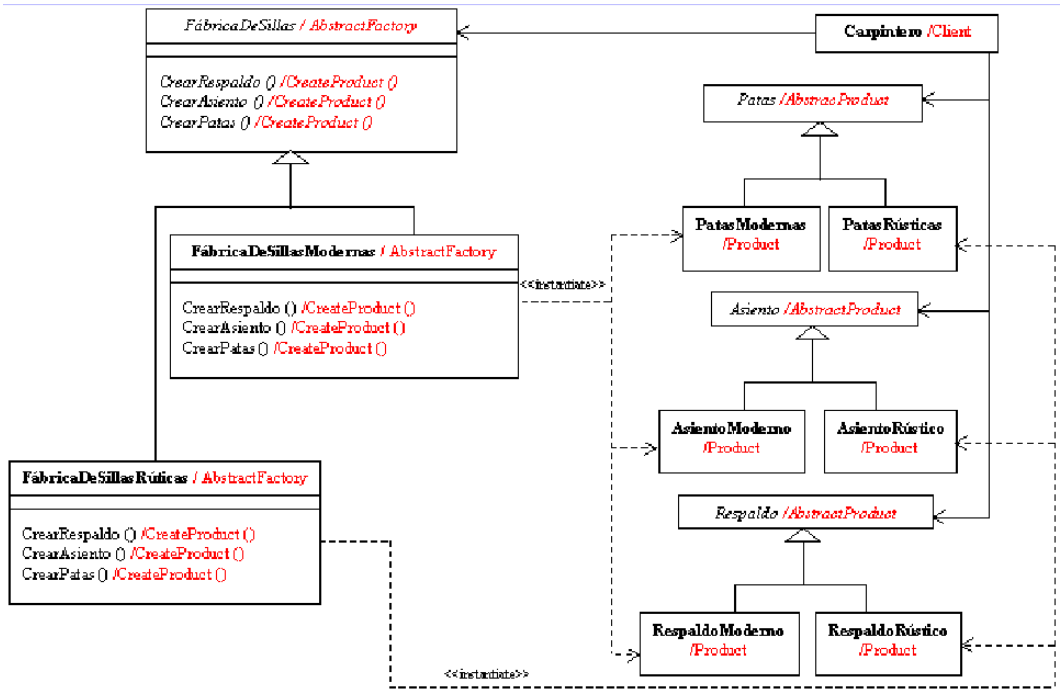

Figura 4. Una instancia del diagrama REP correspondiente al patrón Abstract Factory.

En la figura 4 se muestra una instancia concreta de dicho patrón. En ésta se pueden comprobar las restricciones que impone el patrón abstract factory. Se visualiza el rol-elemento que juega cada elemento de diseño usando un color que identifica el patrón unívocamente. Si el elemento participa en patrones diferentes, éstos pueden especificarse secuencialmente. Esta característica podría visualizarse u ocultarse según convenga mediante una herramienta.

5. Conclusiones y trabajo futuro

Llevar a cabo la especificación precisa de un patrón de diseño es complicado ya que no es fácil dar respuesta a las siguientes preguntas clave:

- ¿Qué aspectos deberían especificarse?, es decir, ¿qué debería formar parte de la esencia de un patrón?. No hay que olvidar que un patrón describe un problema, una solución y las consecuencias de su uso. Siendo conscientes de que la complejidad intrínseca de un patrón es grande, parece ser que los trabajos existentes, incluido el presente, se centran sobre todo en la especificación más o menos precisa de esa solución. De momento, en este trabajo nos hemos centrado en los aspectos estructurales de dicha solución, sin embargo, queda pendiente para posteriores trabajos la especificación de las propiedades de comportamiento.
- ¿Cómo realizar esta especificación?. La respuesta necesariamente dependerá de la respuesta dada a la pregunta anterior. El modelo aquí presentado realiza una especificación precisa y visual de la estructura de un patrón a través de una sencilla extensión a UML, sin necesidad de recurrir a una notación formal como puede ser OCL. El resultado es una notación cercana al diseñador, simple, completa y fácil de aprender, y un intuitivo modelo que trata los patrones de diseño como si fueran "verdaderas" plantillas.

De momento hemos conseguido especificar sólo las restricciones estructurales de una manera visual y precisa, mediante los denominados diagramas REP. Respecto a la especificación de las propiedades de comportamiento, pensamos que es necesario investigar cómo adaptar los diagramas de interacción de UML al nivel de abstracción de un patrón, cómo podría ayudar una extensión de OCL con lógica temporal, etc.

Por las características del modelo presentado, pensamos que el desarrollo de una futura herramienta basada en esta aproximación puede ser de un gran interés.

Referencias

Eden, A. H. (2000). "Precise Specification of Design Patterns and Tool Support in their Application", PhD diss., Department of Computer Science, Tel Aviv University.

Eden, A. H. (2002). "LePUS: A Visual Formalism for Object-Oriented Architectures". The 6th World Conference on Integrated Design and Process Technology. Pasadena, California, June 26-30, 2002.

Eden, A. H., Gil, J. and Yehudai, A. (1997). "Precise Specification and Automatic Application of Design Patterns". In The 12th IEEE International Automated Software Engineering Conference – ASE 1997.

http://www.math.tau.ac.il/~eden/bibliography.html#ase

Eden, A. H., Hirshfeld, Y. and Yehudai, A. (1998) "LePUS – A Declarative Pattern Specification Language". Technical report 326/98, department of computer science, Tel Aviv University. 1998.

http://www.math.tau.ac.il/~eden/bibliography.html#lepus

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). Design Patterns: Elements Reusable of Object-Oriented Software. Addison Wesley Professional Computing Series, Reading, MA.

Isla, J.L. (2002). "A new approach for modelling design patterns with UML". The 12th PHDOOS workshop (ECOOP2002), Málaga, 10-14 june 2002, Spain.

http://www.softlab.ece.ntua.gr/facilities/public/AD/phdoos02/jose.ps

Kent, S. (1997). "Constraint Diagrams: Visualising Invariants in Object-Oriented Models". In Proceedings of OOPSLA'97. ACM Press.

Lauder, A. and Kent, S. (1998). "Precise Visual Specification of Design Patterns". In Proceedings of ECOOP'98. Springer-Verlag.

Le Guennec, A., Sunyé, G. and Jézéquel, J. (2000). "Precise Modeling of Design Patterns". In Proc. UML' 2000. Springer Verlag, LNCS 1939.

Meijers, M. (1996). Tool Support for Object-Oriented Design Patterns. Master's Thesis, INF-SCR-96-28. Utrecht University.

Object Management Group. (2002). OMG Unified Modeling Language Specification V 1.4., http://www.omg.org

Riehle, D. (1996). "Describing and Composing Patterns Using Role Diagrams." WOON '96 (1st International Conference on Object-Orientation in Russia), Conference Proceedings. St. Petersburg Electrotechnical University, 1996. Reprinted in K.-U. Mätzel and Hans-Peter Frei (eds.), Proceedings of The Ubilab Conference '96, Zürich. Germany, Universitätsverlag Konstanz, pp. 137-152.

Sunyé, G., Le Guennec, A. and Jézéquel, J. (2000). "Design Patterns Application in UML". In Proc. European Conference in Object Oriented Programming – ECOOP' 2000. Springer Verlag, LNCS 1850.