Club de Investigación Tecnológica

Patrones de software

Alan Calderón Castro

Agosto 2006

Club de Investigación Tecnológica

Informes publicados

	Informe	Autor	Fecha
1.	Redes de Computadores	Dr. Roberto Sasso	Agosto 1988
2.	Sistemas Expertos	Dr. Claudio Gutiérrez	Enero 1989
3.	Planificación de Sistemas	Dr. René-Pierre Bondu	Abril 1989
4.	Proyectos de Sistemas	Ing. Ignacio Trejos	Setiembre 1989
5.	Bases de Datos	Dr. Carlos González	Diciembre 1989
6.	Escapando de los Sistemas del Ayer	Lic. Pablo Rojas, M.Sc.	Marzo 1990
7.	Aplicaciones Creativas	Dr. Roberto Sasso	Mayo 1990
8.	Calidad de Sistemas	Dr. Ulises Agüero	Octubre 1990
9.	Personal y Organización de Sistemas	KPMG Consultores	Marzo 1991
10.	Sistemas Abiertos	Ing. José Rubinstein, MBA	Octubre 1991
11.	Análisis de la Industria de la TI.	Lic. Roberto Venegas, MBA	Enero 1992
12.	Nuevas Tecnologías de Información	Dr. Roberto Sasso (Editor)	Marzo 1992
13.	Ambientes de Proveedores Múltiples	Lic. Alexis Rodríguez U.	Julio 1992
14.	Planificación y Recuperación de Desastres	Sr. Gerardo Ortuño	Agosto 1992
15.	Diseño de Redes Novell	Ing. David Baruch	Agosto 1993
16.	Minis Vs LANs	Ing. Marvin Campos	Octubre 1993
17.	Intercambio Electrónico de Datos (EDI)	KPMG Consultores	Enero 1995
18.	Sistemas Abiertos de Software	Ing. José Ardón	Abril 1995
19.	Outsourcing de Tecnología de Información	Roxana Murillo, M.Sc.	Julio 1996
20.	Redes Empresariales de Banda Ancha	Ing. Aníbal Mayorga, M.Sc.	Febrero 1997
21.	Comercio Electrónico	Dr. Roberto Sasso Rojas	Abril 1997
22.	Estudio de Opinión Informática	Dr. Freddy Abarca	Julio 1997
23.	Desarrollo de Sistemas Cliente/Servidor	Lic. Édgar Hernández Ing. Luis Martínez	Diciembre 1997
24.	Enfrentando el año 2000. Guía Práctica	Ing. Carlos Gallegos, M.Sc. Dr. Roberto Sasso Ing. Ignacio Trejos, M.Sc.	Mayo 1998
25.	Depósitos de Datos	Beatriz Jiménez, M.Sc. Rafael Avalos, M.Sc.	Noviembre 1998
26	El modelo de objetos: Análisis y Diseño	Ing. Ignacio Trejos, M.Sc. Ing. Antonio Luna	Setiembre 1999
27.	Silicon Valley, 1999	Ing. Mauricio Monge Dr. Roberto Sasso Ing. Ignacio Trejos, M.Sc.	Enero 2000
28.	Calidad de los datos: Un enfoque conceptual	Ing. Lilia Muñoz, M.Sc.	Febrero 2000
29.	El modelo de objetos: Lenguaje de modelaje Unificado (UML)	Ing. Antonio Luna Ing. Ignacio Trejos, M.Sc.	Marzo 2000
30.	Medición de calidad de datos: Un enfoque práctico	Ing. Franco Quirós	Marzo 2000
31.	Seguridad de la información en la era de los negocios digitales	Lic. Édgar Hernández Lic. Marco V. Gámez	Julio 2001
32.	Transformación de aplicaciones legacy	Ing. Declan Good	Agosto 2002
32.	Legacy transformation	Ing. Declan Good	August 2002
33.	Calidad en la especificación de requerimientos	Ing. Javier Rivas	Febrero 2003
34.	Inteligencia de negocios	Lic. José Mayorga	Setiembre 2004
35.	Sistemas colaborativos	Ing. Xinia Robles Lic. Lizette Ramírez, M.Sc.	Octubre 2004
36.	XML: Tecnología y aplicaciones	Dr. José Enrique Araya Ing. Emilia Zeledón	Enero 2005
37.	Procesos de software	Ing. Priscilla Garbanzo, MIS	Setiembre 2005
38.	Patrones de software	Lic. Alan Calderón, M.Sc.	Agosto 2006

Editado y publicado por Rho-Sigma, S.A., a nombre del Club de Investigación Tecnológica.

San José, Costa Rica. Agosto 2006

Resumen Ejecutivo

La producción de software es una actividad intensiva en conocimiento. Sin embargo, en nuestro medio no se ha prestado suficiente atención a la sistematización y organización del conocimiento que desarrollan los ingenieros de software cuando se enfrentan a un dominio de aplicaciones a lo largo de varios años, en distintos proyectos orientados a satisfacer las necesidades de información de clientes específicos.

Hace aproximadamente diez años, varios expertos en ingeniería del software formularon una forma de sistematizar y organizar su conocimiento especializado para facilitar su difusión en la comunidad de ingenieros de software. Los ***patrones*** permiten documentar y organizar de manera efectiva el conocimiento construido en cualquier dominio relevante como por ejemplo el diseño detallado de objetos, el diseño arquitectónico o la programación en algún lenguaje de alto nivel particular.

Este informe contribuye a difundir el concepto de ***patrón*** para que nuestras organizaciones desarrolladoras de software puedan sacar provecho del inmenso acervo de ***patrones*** disponible en textos, artículos y sitios de la Internet. A la vez, se plantea la necesidad de sistematizar el conocimiento específico generado por desarrolladores de software en relación con un dominio de aplicación, por medio de la elaboración y organización ***patrones de análisis de dominio***.

Del autor
Alan Calderón Castro es profesor asociado de la Universidad de Costa Rica. Es licenciado en Ciencias de la Computación e Informática de la Universidad de Costa Rica. Realizó su investigación de tesis de Maestría (Ciencias Cognoscitivas) en el Sistema de Estudios de Posgrado de la Universidad de Costa Rica sobre el tema de organización del conocimiento de patrones en grupos de ingenieros de software.

Agradecimientos
Un sincero y efusivo agradecimiento a Ignacio Trejos por todo el tiempo dedicado a la revisión de este informe. La pertinencia de sus observaciones y comentarios han contribuido a mejorar enormemente la calidad del informe.

Nota editorial
Este informe fue revisado por Antonio Luna, Jimmy Figueroa, Roberto Sasso, Mario Agüero e Ignacio Trejos. La edición final estuvo a cargo de Ignacio Trejos.

Tabla de contenidos

Índice de figuras

Figuras del Apéndice 1

1. Introducción

Es muy probable que, en su trabajo como ingeniero de software, el lector se haya encontrado con temas como la clasificación de cuentas en un sistema contable, o la clasificación de tipos de productos en un sistema de inventario o de catálogo de productos, o la clasificación de puestos en sistemas de información para la administración de recursos humanos o la clasificación de clientes o proveedores en un sistema orientado a facilitar el manejo de la información de los contactos de una empresa. Los más experimentados podrán, probablemente, seguir agregando a la lista "variaciones de un mismo tema". Cualquier persona que de vez en cuando se da el tiempo para reflexionar un poco sobre su práctica profesional se pregunta: "¿habrá algo en común de lo cual yo pueda sacar provecho para mi trabajo cotidiano?". Y la respuesta es obviamente ¡¡SI!! Entre los ejemplos citados hay esquemas de interacción humano-sistema que se repiten, hay modelos de datos similares y diseños detallados de clases que muestran coincidencias. ¿Cómo sintetizar estas similitudes y organizar las diferencias, de tal manera que se pueda sacar provecho para el desarrollo de nuevos sistemas de software? En los últimos doce años, los ingenieros de software han estado documentando ***patrones*** para sacar provecho a esta enorme masa de experiencia acumulada. ¿Va usted a mantenerse al margen de los beneficios de todos estos *patrones*?

En el epígrafe de [Gamma 1995] se cita la siguiente declaración de Tom DeMarco "...este nuevo libro de Gamma, Helm, Johnson y Vlissides promete tener un impacto importante y duradero en la disciplina del diseño de software. Debido a que *Design Patterns* se autopromueve como concerniente a software orientado a objetos únicamente, me temo que los desarrolladores de software que no participan en la comunidad de objetos pueden ignorarlo. Esto sería una lástima. Este libro tiene algo para cualquiera que diseñe software. Todos los diseñadores de software usan patrones, entender mejor las abstracciones de nuestro trabajo sólo nos puede hacer mejores en él."[1].

Este comentario de Tom DeMarco es una de esas ocasiones excepcionales en la historia de la ciencia y la tecnología en que dos generaciones, que representan paradigmas diferentes, en principio inconmensurables como ha señalado Kuhn [Kuhn 1986], logran converger. De alguna manera, lo mejor y lo más representativo del antiguo paradigma se reconoce en lo mejor y más representativo del nuevo, de donde emerge y se sustenta un continuo, al menos convencional, en la historia de la ciencia y la tecnología. Y es que, como suele suceder, el paradigma orientado a objetos replantea algunos de los problemas, temas, prácticas y conceptos centrales del "paradigma estructurado"[2] y a la vez da lugar a nuevos temas, problemas, prácticas y conceptos: este el caso precisamente de los *patrones* y los *lenguajes de patrones*. Así es como "*Design Patterns*" (ver [GoF]) constituye un punto culminante en el paradigma orientado a objetos, de la misma forma como las obras de Tom DeMarco [DeMarco 1978] y Yourdon y Constantine [Yourdon 1978] han sido hitos para el antiguo paradigma. Según el comentario de DeMarco, el punto crucial que une ambas tradiciones es que "...Todos los diseñadores de software usan patrones...", no importa si trabajan desde el paradigma estructurado o desde el nuevo paradigma orientado a objetos, DeMarco vislumbra que los diseñadores de software podrán mejorar en su trabajo si logran comprender y reutilizar las abstracciones que usan como herramientas en su trabajo cotidiano. El gran

[1] Tom DeMarco, IEEE *Software*.
[2] Análisis y diseño estructurado tal como fuera elaborado por Tom DeMarco, Edward Yourdon, Larry Constantine y otros.

mérito de [GoF] y otros importantes catálogos de patrones que se citan en este informe es haber planteado los patrones y lenguajes de patrones como un tema central para el diseño de software. Los patrones y lenguajes de patrones son las mejores herramientas que los desarrolladores de software han logrado encontrar hasta la fecha para obtener el máximo provecho del conocimiento elaborado a lo largo de muchos años de experiencia. En la actualidad, son considerados la mejor estrategia para sistematizar el conocimiento de expertos en diferentes dominios de aplicación y en las distintas áreas del proceso de desarrollo de software.

Este informe pretende resumir lo más relevante del estado del arte en el tema de patrones y lenguajes de patrones desde una perspectiva puramente práctica, es decir, con la intención de ser una guía útil para el ingeniero de software en cualquier empresa desarrolladora de software. El informe inicia con la caracterización precisa del concepto de patrón y los beneficios generales que se pueden derivar del uso de patrones en el proceso de desarrollo de software. Luego se describen más detalladamente los patrones de análisis de dominio, los patrones arquitectónicos y los patrones de diseño así como los beneficios particulares en cada caso. Los ejemplos que se utilizan en la segunda parte, se han desarrollado más ampliamente en el Apéndice 1. En la tercera parte del informe se caracteriza el concepto general de "lenguaje de patrones" y se contextualiza el tema con el concepto de "lenguaje de patrones de análisis de dominio" a fin de evidenciar su valor práctico en un marco de desarrollo de software orientado a "familias de productos de software" o "software product lines", según [Clements 2002]. En esta tercera sección se muestra, mediante un ejemplo, cómo se puede estructurar un "lexicón de patrones" para que sirva como herramienta de trabajo en la administración del conocimiento especializado de "lenguajes de patrones de análisis de dominio" en una empresa desarrolladora de software. Este constituye el aporte original de este informe al estado del arte. Finalmente se proveen algunas recomendaciones generales para el aprovechamiento de los patrones y lenguajes de patrones en una empresa desarrolladora de software. El Apéndice 2 incluye una lista de sitios en Internet con contenidos relevantes.

2. ¿Qué se entiende por "patrón"?

Siempre resulta interesante el hecho de que el concepto de patrón se originara en el campo de la arquitectura. Es Christopher Alexander quien por primera vez utiliza el término en forma similar a como luego ha sido usado ampliamente en la ingeniería del software. En uno de sus principales libros, nos dice que "*Un patrón describe primero un problema que ocurre una y otra vez en nuestro entorno, y a continuación describe el núcleo de la solución a dicho problema, de tal manera que usted puede usar la solución millones de veces sin repetir la solución específica una sola vez.*" [Alexander 1979]. En el contexto de la ingeniería del software, es muy posible que el concepto haya sido usado por primera vez por Johnson [Johnson 1992] y desde entonces ha evolucionado adquiriendo connotaciones propias de nuestro campo durante aproximadamente trece años. Por esta razón conviene complementar esta caracterización con los aportes de especialistas en patrones usados en distintas áreas de la ingeniería del software:

1. En el conocidísimo y famoso libro de [GoF], aparte del texto citado de Alexander, se acota que un patrón tiene cuatro elementos esenciales:
    a. *Nombre*: es una palabra o frase corta (dos o tres palabras) que simboliza lo fundamental de un patrón, el tipo de problemas que resuelve y las características generales de la solución.
    b. *Problema*: caracteriza el tipo de problemas y el contexto asociado a que apunta el patrón.
    c. *Solución*: describe los aspectos fundamentales de la solución, incluyendo objetos, asociaciones, así como responsabilidades de los objetos[3] y colaboraciones entre ellos.
    d. *Consecuencias*: describe los efectos generales sobre un diseño de software al adaptar la solución a un contexto específico, cuáles son sus beneficios e inconvenientes desde el punto de vista de factores como el desempeño, la flexibilidad, la reutilizabilidad, la extensibilidad (ampliación de las funciones) y la portabilidad de un sistema de software.
2. Los aportes de Fowler a este tema son muy apreciados y conviene destacar que su caracterización de "patrón" es intencionalmente simple: "*Un patrón es una idea que ha sido útil en algún contexto práctico y probablemente será útil en otros.*" [Fowler 1997]. Este autor argumenta que prefiere esta caracterización simple y poco restrictiva para mantenerse lo más cerca posible de la intencionalidad original de los patrones. Agrega que los patrones pueden presentarse en múltiples formas[4] que reflejan especializaciones útiles para la clase de patrones que cada uno representa[5].
3. Los catálogos [POSA1] y [POSA2] han influido fuertemente en el desarrollo del concepto de patrón: "*Un patrón para arquitectura de software describe un problema de diseño recurrente que surge en contextos de diseño específicos y presenta un esquema general de solución cuyas conveniencias han sido demostradas. El esquema de la solución se especifica describiendo los componentes constituyentes, sus responsabilidades y conexiones, así como la forma en que colaboran*" (pág. 8 de [POSA1]). Aunque esta caracterización se enmarca en el contexto del diseño

[3] Ver [Trejos 1999] y [Luna 2000] para refrescar conceptos sobre orientación a objetos.
[4] Se puede interpretar aquí que el autor se refiere a las plantillas o esquemas de documento usados para especificar un patrón.
[5] Como veremos más adelante existen varias clases de patrones asociadas con distintas áreas del proceso de desarrollo de software lo que ha llevado a generar distintas formas de documentar los patrones.

arquitectónico de software, sin embargo, su significado puede ser generalizado abstrayendo el carácter técnico de los términos “arquitectura de software” y “componente”.

En este documento se usará la caracterización de [GoF] agregando un quinto elemento que es fundamental: *interconexiones*. “Ningún patrón es una isla” nos dice Alexander en [Alexander 1979], lo cual significa básicamente que ningún patrón se utiliza en forma aislada y por ende ningún patrón puede ser comprendido sino en el contexto de la red de patrones en que ha sido especificado. Por esta razón se puede verificar que en todos los catálogos de patrones, la especificación de cada patrón siempre incluye referencias a otros patrones.

El concepto de patrón no puede ser bien presentado si no se complementa con las consideraciones generales sobre su utilidad: ¿para qué sirve un patrón? La respuesta a esta pregunta introduce necesariamente el tema de las distintas clases o categorías de patrones que poco a poco han ido surgiendo en el campo de la ingeniería del software. También hace falta aportar ejemplos, pero de nuevo aquí surge la necesidad de analizar las distintas clases de patrones, porque de lo contrario se correría el riesgo de sesgar la construcción del concepto si solo se presentase un ejemplo de alguna de las clases de patrones existentes. Estos son los temas centrales de la próxima sección.

3. Uso de patrones y sus beneficios

Como resultado de un estudio realizado a través de la Internet entre el año 2000 y el 2001, Czichy [Czichy 2001] concluye que en la industria se están usando patrones para distintas áreas del proceso de desarrollo de software. La figura #1, tomada directamente de la versión electrónica del estudio referido, muestra estas áreas e inclusive indica su uso ya establecido en el ámbito educativo.

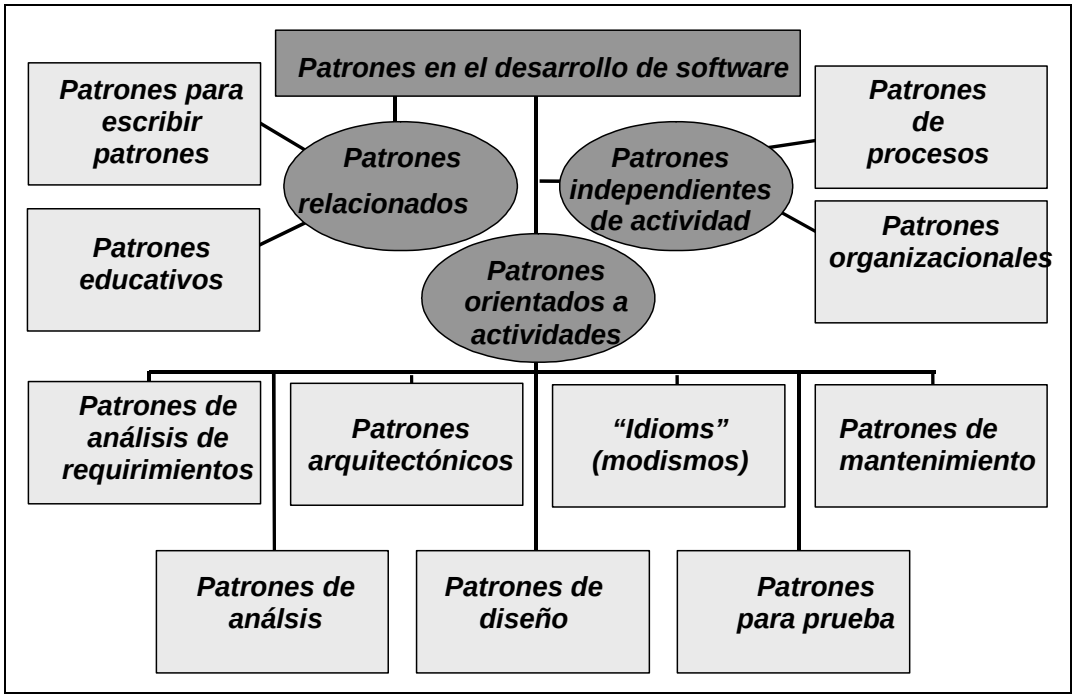

Figura #1: Uso comprobado de patrones en distintas áreas del desarrollo de software. Tomada de [Czichy 2001].

Obsérvese que los tres procesos clásicos de la ingeniería del software (desarrollo, administración y aseguramiento de la calidad) han sido abarcados, al menos parcialmente, en el gráfico de Czichy. A la fecha, es de esperar que el espectro de actividades cubiertas por los patrones se haya ampliado aún más. Por ejemplo, algunos autores han publicado patrones relacionados con el diseño de la interacción humano-sistema (véase [PI] y [IDP]). La mayoría de las categorías establecidas en el gráfico pertenecen a actividades del proceso de desarrollo, lo cual refleja bien lo que ha sucedido en este campo de la ingeniería del software. Para efectos prácticos, los patrones que se usan en distintas actividades del proceso de construcción son, por decirlo así, los patrones por antonomasia. Este informe, se centrará en esta categoría general de patrones por razones de espacio y de estrategia didáctica.

Se puede citar una serie de razones generales en respuesta a la pregunta "¿para qué sirven los patrones?", pero no se puede omitir que parte de la respuesta debe aludir al tipo de actividad al que esté orientado cada patrón en particular. A partir de la realización de la primera conferencia internacional sobre patrones "Pattern Languages of Programs" en 1994 (véase [PLoP] y [PHP]), se han venido realizando conferencias anualmente en varios países. En la actualidad, se realizan casi todos los años conferencias en E.E.U.U. en Illinois y Texas, en

Europa en Dinamarca y Alemania, en Australia y en Brasil. Esto ha provocado una proliferación de patrones que hace simplemente imposible pretender abarcar todas las áreas en este informe. Una estrategia didáctica mejor consiste en abarcar unas pocas áreas y de ellas unos pocos patrones que muestren claramente la naturaleza y uso de éstos, así como su potencial desde el punto de vista de las organizaciones que desarrollan software.

A continuación se describen beneficios generales del uso de patrones en el proceso de desarrollo de software, para luego extender este análisis haciendo un recorrido corto por algunas de las actividades de dicho proceso citando ejemplos particulares que se desarrollan más ampliamente en el Apéndice 1 de este informe. En dicho apéndice se ha incluido el análisis y diseño arquitectónico de un pequeño sistema de información empresarial con la intención de mostrar con detalle el uso de algunos pocos patrones.

3.1 Beneficios generales del uso de patrones

Son muchos los beneficios generales del uso de patrones en el proceso de desarrollo de software que se reportan en la literatura especializada. La siguiente lista ha sido sintetizada a partir de los beneficios más reiterados en [Alur 2001], [Bruegge 2002], [Coad 1997], [Czichy 2001], [Fowler 1997], [GoF], [Hay 1996], [Larman 1998], [POSA1], [POSA2], [Pree 1995] y se complementa con resultados de la investigación expuesta en [Calderón 2003].

1. *Mejora la comunicación entre grupos de ingenieros de software*: Lo decisivo de una comunicación efectiva entre los desarrolladores para el éxito de un proyecto de construcción de software ha sido plenamente justificado por muchos autores, entre ellos Sommerville [Sommerville 2002]. Los patrones permiten una comunicación más efectiva al ampliar el lenguaje común para hablar sobre diseño y resolver problemas de diseño colaborativamente. Cuando un grupo de desarrolladores conoce suficientemente un conjunto de patrones y sus interconextiones ya no necesitan discutir en términos de clases y métodos las alternativas de solución a un problema, sino que lo pueden hacer en términos de patrones y sus relaciones, lo cual hace más efectiva la comunicación.
2. *Facilita un mejor manejo de la complejidad en el diseño*: “Los patrones de diseño proveen un vocabulario común a los diseñadores para que lo usen en la comunicación, documentación y exploración de alternativas de diseño. Los patrones de diseño hacen que un sistema se vea menos complejo al permitirle a usted, como ingeniero de software, hablar sobre un sistema en un nivel más alto de abstracción que aquél que le permite la notación de diseño o el lenguaje de programación. Los patrones de diseño permiten elevar el nivel en el cual los ingenieros de software discuten o diseñan software." (Traducción de pág. 352 de [GoF]). A diferencia del beneficio anterior, aquí el énfasis está en la abstracción que permite el uso de patrones, lo cual facilita el diseño de sistemas complejos.
3. *Facilita el diseño colaborativo*: Debido a los dos beneficios anteriores (mejor comunicación y más alto nivel de abstracción), la resolución de problemas de diseño en forma colaborativa se ve facilitada.
4. *Facilita el aprendizaje del uso apropiado de “frameworks”*: En general, los patrones promueven buenas prácticas en el uso de tecnologías de desarrollo. Por ejemplo, el catálogo de patrones de [GoF] bien puede verse como un conjunto de buenas prácticas de diseño de software orientado a objetos. Este paradigma de desarrollo se sustenta en la maduración de una tecnología de desarrollo específica. De hecho en algunos pocos

patrones de [GoF] la solución genérica aportada no se puede deslindar del uso de C++, aunque la mayoría es igualmente aplicable en Java u otros lenguajes de programación orientada a objetos, lo cierto es que no tendrían casi ningún sentido para un proyecto de desarrollo en el que se use una tecnología basada en programación declarativa o funcional. Por esta razón, no es de extrañar que algunos catálogos de patrones estén orientados a promover buenas prácticas de programación de tecnologías aún más específicas. Es el caso precisamente de [Alur 2001] que es un catálogo de patrones para promover buenas prácticas de desarrollo de software entre los desarrolladores que usan la plataforma J2EE de Sun Microsystems. Conforme las plataformas de desarrollo se basan en "frameworks" más complejos, la curva de aprendizaje es el reto principal tanto para la empresa que intenta sacar provecho de las tecnologías más recientes como para el desarrollador individual que hace un esfuerzo por mantenerse competitivo. Una documentación basada en patrones típicos de uso de un "framework" puede facilitar enormemente su aprendizaje.

5. *Orienta la verificación como parte del control de calidad*: El modelo de análisis, el diseño arquitectónico, el diseño detallado y hasta el código de un sistema son ejemplos de subproductos del proceso de desarrollo de software que pueden ser sometidos a verificación a fin de garantizar su calidad. Una actividad de verificación de este tipo, podría orientarse con base en patrones. Los patrones documentan esquemas bien conocidos y probados que aseguran muchos atributos de calidad. Al contrastar una solución específica contra la solución genérica de un patrón atingente, es posible identificar aspectos a mejorar o al menos concluir que no carece de características de calidad fundamentales reveladas por el patrón. Por ejemplo, una característica fundamental de "Multicapas" de [POSA1] consiste en que las capas inferiores (asociadas con funciones de más bajo nivel) nunca invocan servicios de capas superiores. Esta carácterística garantiza portabilidad, modificabilidad y en general la posibilidad de evolución independiente de las capas. Al verificar el diseño arquitectónico de un sistema, el uso de "Multicapas" podría revelar que esta característica no se respeta completamente y de esta manera el contraste motivaría una mejora.
6. *Facilita la sistematización de conocimientos de diseño y de resolución de problemas*: El conocimiento básico de diseño detallado orientado a objetos incluye una gran cantidad de principios y heurísticas que con frecuencia permanecen implícitas o no se contextualizan apropiadamente en los textos de ingeniería del software orientado a objetos. Tan solo como ejemplo, considérese el principio de "Minimizar el acoplamiento". Este principio, que originalmente fuera formulado por Constantine [Constantine 1979] dentro del paradigma de desarrollo de software estructurado, y que luego ha sido reinterpretado en el contexto del paradigma de objetos (véase por ejemplo [Bruegge 2002]), puede especificarse simplemente como minimizar la cantidad de conexiones entre componentes o disminuir la dependencia entre componentes usando conexiones más débiles. A falta de mayor contextualización, es posible que la aplicación del principio derive en una práctica "a posteriori", en la que se trata de "medir" la cantidad de acoplamiento entre un conjunto de componentes de un sistema para luego intentar disminuir dicha cantidad o buscar acoplamientos más débiles. En contraste, el patrón "Fachada" de [GoF] establece, "a priori" que para minimizar la dependencia entre componentes o subsistemas conviene incorporar "del lado del servidor" (bajo una dinámica de interacción "cliente/servidor") o en "ambos lados" (bajo una dinámica "par a par") un objeto cuya única función sea simplificar la interacción y encapsular la estructura interna de un componente a sus componentes "cliente". En otro caso, el patrón "Adaptador" establece cómo un componente puede usar otro no previsto originalmente interponiendo un objeto

intermediario que transforme la interfaz provista por el nuevo según las necesidades de acoplamiento del componente cliente, inicialmente construido con la intención de usar los servicios de un componente dado. Al desacoplar el componente cliente del componente servidor, el "Adaptador" se convierte en un intermediario que facilita la evolución del sistema, permitiendo intercambiar el componente servidor por versiones más recientes o más apropiadas, sin que el componente cliente deba ser modificado. Tenemos aquí dos contextos específicos en que el principio de minimización del acoplamiento se aplica. Al estudiar y documentar estos contextos y cómo se aplica el principio, ambos patrones facilitan su puesta en práctica. El principio, en su formulación sencilla, es importante, pero muchas veces insuficiente para derivar una práctica pro-activa de diseño detallado de objetos.

7. *Facilita la adquisición de conocimientos sobre diseño y técnicas de resolución de problemas*: Se puede ver de la presentación del beneficio anterior, que los patrones constituyen pericia explícita y sistematizada.El análisis de contextos, problemas y soluciones típicamente es producto de una amplia experiencia en diseño de software o en actividades de consultoría por parte de los autores de patrones. Por ende, los patrones son herramientas fundamentales de aprendizaje, tanto para el aprendiz, como para el desarrollador profesional que constantemente debe enfrentar el reto de apropiarse de nuevas tecnologías de desarrollo (patrones para documentar el buen uso de "frameworks") o comprender nuevos dominios de aplicación (dominios técnológicos, como es el caso de la programación concurrente o "dominios de negocio", como podría ser el caso de "aplicaciones para el mercado de valores"). Los patrones pueden ser usados para sistematizar conocimiento en cualquiera de estos casos y esto no solo es relevante para las instituciones educativas, sino también para la empresa desarrolladora de software que no puede soslayar el hecho de que, en el aprendizaje de sus desarrolladores de mayor experiencia, hay un capital de conocimiento irrenunciable que se destaca ante el peligro que representa la rotación de personal altamente especializado.

El análisis de otros beneficios conlleva enfocarse en una área o categoría específica según el diagrama de la figura #1 [Czichy 2001]. Por ahora nos contentamos con brindar una vista panorámica del estado del arte en este campo y motivar al lector a usar y a producir patrones. A continuación se profundiza la presentación sobre para qué sirven los patrones, los beneficios de usarlos y sus características, enfocándose en tres actividades típicas del proceso de desarrollo, a saber: el modelado de entidades y relaciones en el análisis, el diseño arquitectónico y el diseño detallado de objetos. Esto permite a la vez presentar ejemplos concretos de patrones cuya descripción más amplia podrá encontrar el lector en el apéndice.

3.2 Características y beneficios del uso de patrones de análisis

Fowler es uno de los autores más reconocidos en relación con el tema de patrones de análisis. El trabajo de este autor comienza con [Fowler 1997] y continúa en la actualidad (véase [Fowler]). De hecho Hashler [Hashler] le atribuye el término "patrón de análisis". Otros autores que han publicado catálogos completos de patrones de análisis son Coad ([Coad 1992] y [Coad 1997]), Hay [Hay 1996], Silverston [Silverston 1996], Eriksson [Ericksson 2000]. Por otro lado, con frecuencia aparecen patrones de análisis en las memorias de las conferencias PLoP, por ejemplo: [Fernández 2000], [Vaccare 1999], [Fernández 1999a], [Fernández 1999b], [Gaertner 1999], [Grand 1999]. Todos los patrones publicados en los catálogos y artículos referidos pertenecen al dominio de los sistemas de información empresarial. Aunque hasta el momento este dominio ha sido el objetivo principal de los

escritores de patrones de análisis, es de esperar que poco a poco otros dominios como por ejemplo "bioinformática", informática educativa, informática jurídica y "agromática" sean motivo de sistematización del conocimiento por parte de ingenieros de software dedicados a tales dominios de aplicación.

La diversidad entre los patrones de análisis es enorme si se compara con la diversidad entre los patrones de diseño detallado o patrones de diseño arquitectónico. De hecho mientras el catálogo de [GoF] es un hito que ha sido tomado como punto de referencia por los escritores de catálogos de patrones de diseño detallado y de diseño arquitectónico, entre los escritores de patrones de análisis no se ha llegado a un acuerdo similar. Persiste en este ámbito la diversidad de tópicos o categorías de patrones, la diversidad de esquemas de especificación de patrones (patrones para escribir patrones según la figura #1 de Czichy) y todavía en la actualidad se discute sobre cómo lograr patrones de análisis "estables" (véase por ejemplo [Hamza 2002]), es decir cuyo ámbito de aplicación sea suficientemente amplio y que no deban ser redefinidos con frecuencia al aplicarlos, tal como sucede con los patrones de [GoF].

La figura #2 tomada de [Hashler] muestra en sombreado las áreas en que los patrones de análisis pueden ser usados. Este autor indica claramente un área con bordes difusos que se ubica entre la estructuración del problema concreto y el diseño centrado en la solución que bien podría incluir parte del diseño arquitectónico y parte del diseño detallado, pasando por la actividad de análisis como convencionalmente se le conoce. Así por ejemplo, catálogos como el de Hay [Hay 1996] y Silverston [Silverston 1996] se concentran en el modelado de entidades y relaciones, definitivamente crucial en el desarrollo de sistemas de información empresarial. Fowler [Folwer 1997], Ericksson [Ericksson 2000] y Coad [Coad 1997] proveen distintos tipos de modelos estáticos y modelos dinámicos (según se entiende esto en el UML, véase [Luna 2000]) para orientar el análisis de sistemas de información empresarial. Modelos similares a los que aparecen en [Fowler 1997] y [Ericksson 2000] se pueden encontrar en las pequeñas colecciones de patrones que se publican en las memorias de las conferencias PLoP. Estos consideran distintas áreas típicas de los sistemas de información empresarial, enfocándose algunas veces en ámbitos más restringidos como la administración de activos en pequeñas y medianas empresas (es el caso de [Vaccare 1999]). En estos trabajos, los modelos estáticos planteados pueden ser útiles para la elaboración del modelo conceptual de objetos (o el modelo entidad relación en un análisis más tradicional), el modelo de datos, el modelo de clases de análisis (si se sigue una estrategia más abstracta o generalista según los lineamientos de la OMG MDA®, "model-driven architecture" de OMG, véase [OMG] y específicamente [OMG-MDA]) o el modelo de clases del diseño detallado[6]. Los modelos dinámicos pueden ser útiles para sugerir características básicas del diseño arquitectónico, como el flujo de control general o el manejo de la concurrencia.

[6] Cabe destacar que la diferencia fundamental entre ambos modelos de clases consiste en que el modelo de clases del diseño detallado debe incorporar las restricciones asociadas a la tecnología de desarrollo. escogida, mientras que en el modelo de clases del análisis prevalece el punto de vista del análisis del problema abstrayéndose todavía de la solución misma.

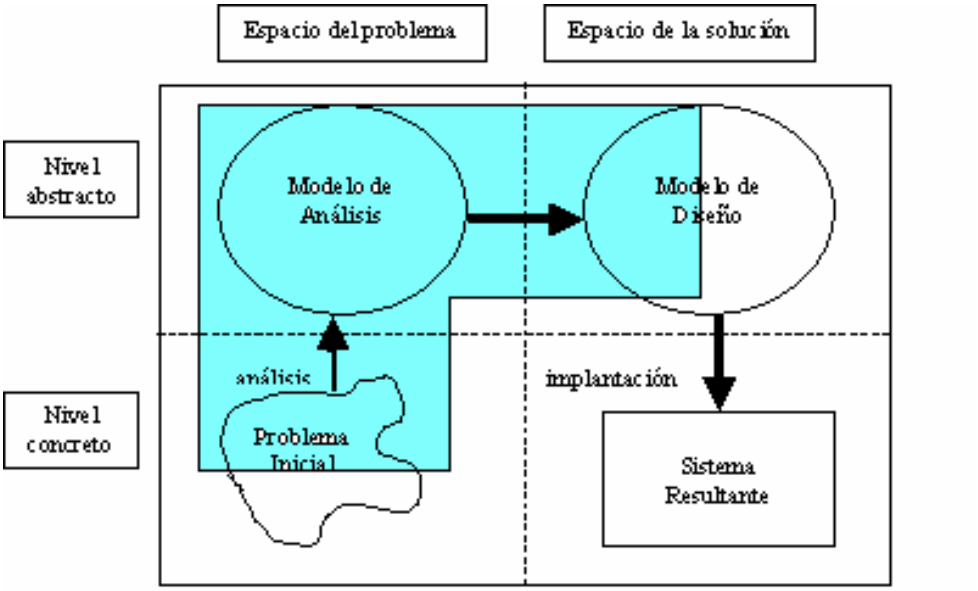

Figura #2: Uso de patrones de análisis en distintas áreas "cercanas" al análisis orientado a objetos. Tomada de [Hashler].

En el contexto de la construcción del modelo conceptual de objetos del problema que se describe en el Apéndice 1, se utilizan varios patrones de análisis de Fowler [Fowler 1997], Silverston [Silverston 1996] y Hay [Hay 1996]. A manera de ejemplo de lo que es un patrón de análisis considérese "Accountability" según [Fowler 1997]. La intención de este patrón es proveer un modelo muy general para representar relaciones entre personas, considerando de manera especial que las relaciones entre personas físicas o jurídicas en sí pueden cambiar con el tiempo y a la vez suelen ser muy variadas. El problema subyacente consiste en cómo crear un modelo conceptual suficientemente flexible que se adapte fácilmente a la evolución de las relaciones entre empresas y suficientemente general que permita representar una inmensa variedad de relaciones entre empresas (y aún más entre personas físicas o jurídicas) que pueden darse. La solución que aporta Fowler aparece en la figura #3. Este autor no incluye mayores detalles sobre los atributos de los objetos que considera en los modelos de sus patrones, pero su trabajo se complementa muy bien con los de Silverston [Silverston 1996] y Hay [Hay 1996] que sí aportan atributos y además permiten enriquecer un modelo con otros conceptos como "Tipo de Estado" y "Tipo de Prioridad" de una relación entre empresas. En la figura #4 se muestra cómo se puede aplicar este patrón a una situación específica como es el caso del ejemplo que se desarrolla en el Apéndice 1.

Para que el modelo conceptual absorba fácilmente nuevas relaciones entre personas existentes Fowler introduce el concepto de "Relación", así cada nueva relación entre dos personas específicas se representará en el sistema como una nueva instancia de este concepto. Para incluir nuevas personas, sería suficiente con el concepto de "Persona", pero el grado de variabilidad a que se enfrenta este modelo exige poder introducir fácilmente nuevos tipos de personas. Por lo tanto, Fowler introduce el concepto de "Clase de Persona". Igualmente, "Relación" sería suficiente para incluir nuevas relaciones entre personas, pero el grado de variabilidad exige poder incluir fácilmente nuevos tipos de relaciones entre personas, por tanto se introduce "Clase de Relación".

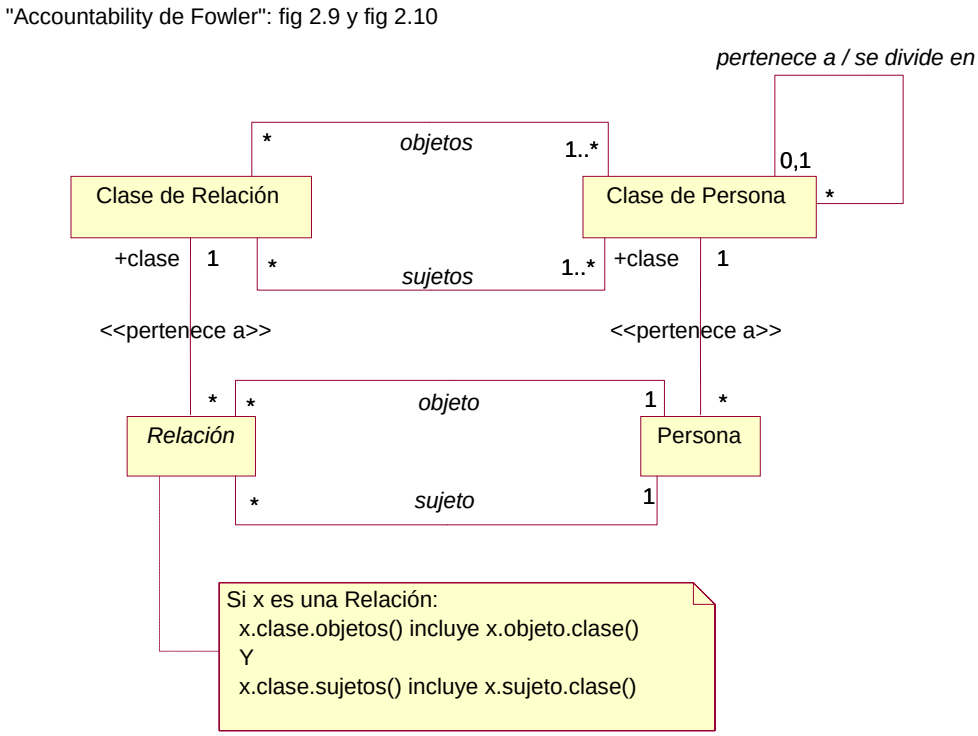

Figura #3: "Accountability" de Fowler [Fowler 1997].

Al plantear un modelo tan general y flexible, surge la necesidad de incluir reglas que garanticen su coherencia, por tanto Fowler introduce las restricciones asociadas a "Relación". Dada una relación específica X entre dos "Personas" A y B, A asume el papel de "objeto" y B el de "sujeto". Entonces para asegurar coherencia en los datos, la clase a que pertenece la relación X debe permitir como "objetos" instancias de la clase a que pertenece A, lo cual significa que la clase a que pertenece A debe estar asociada a la clase de la relación X por medio de la asociación "objetos". Además, la clase a que pertenece la relación específica X debe admitir como "sujetos" instancias de la clase a que pertenece B, lo cual significa que la clase a pertenece B debe estar asociada a la clase de la relación X por medio de la asociación "sujetos". Si X pertenece a la "Clase de Relación" "Cliente de", A es "Alimentos ACME" (que pertenece a "Proveedores de Alimentos") y B es "Maquinaria PRESTON" (que pertenece a "Proveedores de Maquinaria Industrial"). Entonces "Cliente de" debe admitir como "objetos" a instancias de la clase "Proveedores de Alimentos" y como "sujetos" a instancias de la clase "Proveedores de Maquinaria Industrial" lo que significa que "Cliente de" debe estar asociado con "Proveedores de Alimentos" y "Proveedores de Maquinaria Industrial" por medio de "objetos" y "sujetos" respectivamente.

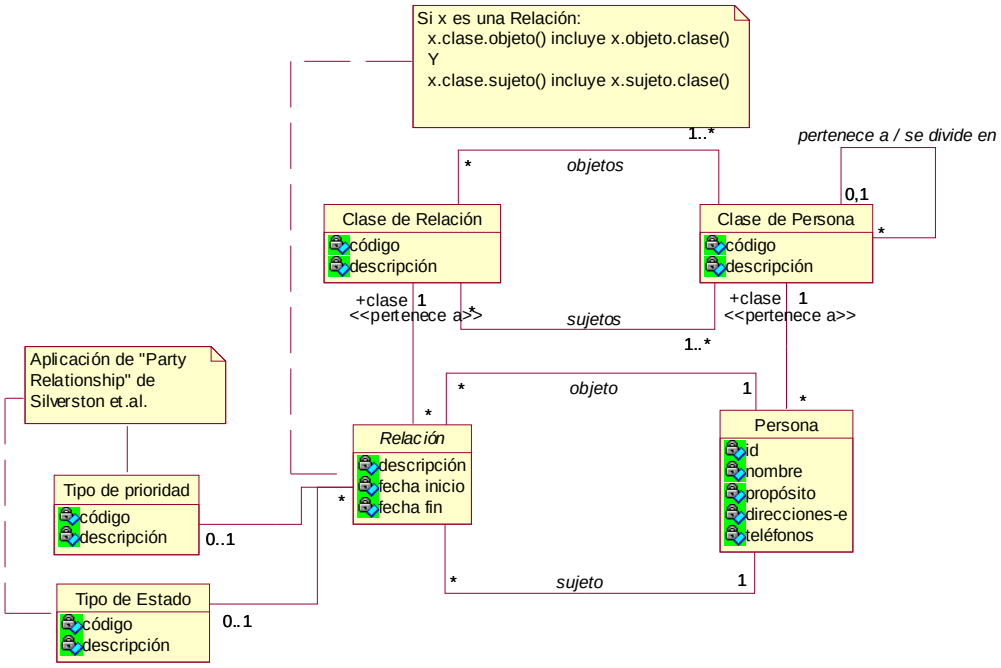

Figura #4: Aplicación de "Accountability" de Fowler [Fowler 1997].

En términos generales se puede afirmar que los beneficios específicos de usar patrones de análisis son:

1. Agilización del análisis en la medida en que los desarrolladores no deben empezar desde cero ("from scratch") sino que inician el trabajo de análisis con modelos plausibles, cuya utilidad de hecho ha sido demostrada, para luego continuar su refinamiento en conversaciones con usuarios, clientes o expertos en el dominio.
2. Reducción del riesgo de error en el análisis debido a que la contextualización de los patrones típicamente incluye aspectos clave del análisis que podrían ser omitidos por un analista poco experimentado. Además, la utilidad de la solución genérica aportada por los patrones ha sido demostrada en otras situaciones, de lo cual al menos se pueden derivar sugerencias atinadas para el modelado de una situación específica. Cabe enfatizar que el riesgo de cometer errores en el análisis es uno de los más importantes en una estrategia de administración del riesgo por cuanto está bien documentado el hecho de que la corrección posterior de estos errores generalmente conlleva costos muy altos.
3. Se facilita la comunicación entre desarrolladores y usuarios, clientes o expertos en el dominio, por cuanto los patrones sustentan un lenguaje básico[7] para discutir sobre el dominio de aplicación.
4. Se facilita la verificación de los artefactos del análisis al contarse con modelos genéricos y de utilidad demostrada contra los cuales se puede contrastar soluciones específicas que surgen de un contexto limitado de análisis, enfocado en las necesidades de una organización específica.

[7] El tema de "lenguajes de patrones" se trata en la sección "Patrones y sistematización del conocimiento para la competitividad".

Es importante señalar que todos estos beneficios adquieren particular relevancia cuando el equipo de analistas no posee mucha experiencia en un dominio de aplicación determinado o simplemente se trata de analistas novatos. El riesgo de introducir defectos durante el análisis, los problemas de comunicación y hasta la "parálisis por análisis" se incrementan sustancialmente cuando se trabaja con analistas poco experimentados. Los patrones de análisis permiten a los desarrolladores incrementar rápidamente sus conocimientos sobre un dominio particular y así mitigar estos riesgos.

Por otro lado, se puede argumentar sobre los beneficios de elaborar patrones de análisis para una empresa desarrolladora de software; este es el tema de la sección "Patrones y sistematización del conocimiento para la competitividad". En la próxima sección se tratan los patrones arquitectónicos o "estilos arquitectónicos".

3.3 Características y beneficios de los patrones de diseño arquitectónico

Los patrones arquitectónicos o estilos arquitectónicos constituyen una categoría de patrones notoriamente más pequeña que la de los patrones de análisis y la de los patrones de diseño. Quizás es debido a esto y a la importancia que tradicionalmente se ha adjudicado al diseño arquitectónico, que están al alcance de los ingenieros de software dos o tres catálogos que sistematizan casi exhaustivamente el conocimiento concerniente a esta actividad del proceso de desarrollo de software. Tres referencias importantes son [POSA1], [POSA2] y [Fowler 2002]. Ya se ha citado en este informe técnico la caracterización de patrón de arquitectura de [POSA1], sin embargo cabe aquí citar aspectos específicos que han sido documentados en dicho catálogo:

1. "*Los patrones constituyen un medio para documentar arquitecturas de software*. Estos permiten describir la visión que usted tiene en mente cuando diseña un sistema de software. Esto permite a otros evitar destruir esta visión cuando deben modificar o extender la arquitectura original, o cuando modifican el código del sistema. Por ejemplo, si usted sabe que un sistema está estructurado según el patrón 'Modelo-Vista-Controlador', entonces usted también sabe cómo debe extenderlo con una función nueva: mantenga el núcleo de la funcionalidad separada del manejo de la entrada de datos del usuario y del despliegue de información" (pág. 7 de [POSA1]).
2. *Los patrones facilitan la construcción de software con propiedades definidas*. Los patrones proveen un esqueleto de comportamiento funcional y de esta manera ayudan a implantar la funcionalidad de su aplicación. Por ejemplo, existen patrones para mantener la consistencia entre componentes cooperantes y para proveer comunicación transparente 'par-a-par' entre procesos. Además, los patrones logran tratar explícitamente requerimientos no-funcionales para los sistemas de software, tales como modificabilidad, confiabilidad, facilidad de prueba o reusabilidad." (pág. 7 de [POSA1]).
3. "*Los patrones le ayudan a construir arquitecturas de software complejas y heterogéneas*. Cada patrón provee un conjunto predefinido de componentes, funciones asociadas y conexiones entre ellos. Un patrón puede ser usado para especificar aspectos particulares de estructuras de software concretas." (pág. 7 de [POSA1]).

En el ejemplo que se desarrolla en el apéndice se aplican tres patrones arquitectónicos: "Multicapas", "Broker" y "Reflexión" de [POSA1]. Dado que "Multicapas" es bien

conocido, se explica a continuación brevemente el patrón “Broker” y cómo ha sido aplicado al ejemplo del apéndice. La aplicación de “Reflexión” es más fácil de apreciar en el modelo de clases de análisis.

Sobre la intencionalidad de “Broker” se ha señalado que “...*Broker* puede ser usado para estructurar sistemas distribuidos de software con componentes desacoplados que interactúan por medio de invocaciones de servicios remotos. Un componente de tipo ‘broker’ es responsable de coordinar la comunicación, como por ejemplo pasando solicitudes de servicios, así como también transmitiendo los resultados y excepciones generadas” (pág. 99 de [POSA1]). Por otro lado, Gomaa señala “Un ‘broker’ es un intermediario en las interacciones entre clientes y servidores. Los servidores registran ante el ‘broker’ los servicios que proveen. Los clientes pueden luego solicitar estos servicios vía el ‘broker’. El ‘broker’ también proporciona **transparencia de localización**, lo cual significa que si el componente de servidor es trasladado a una localización diferente, los clientes no tienen que conocer del cambio y sólo el ‘broker’ debe reconocerlo” (pág. 260 de [Gomaa 2004]). En la figura #5 se muestra la aplicación de “Broker” al caso de estudio del Apéndice 1.

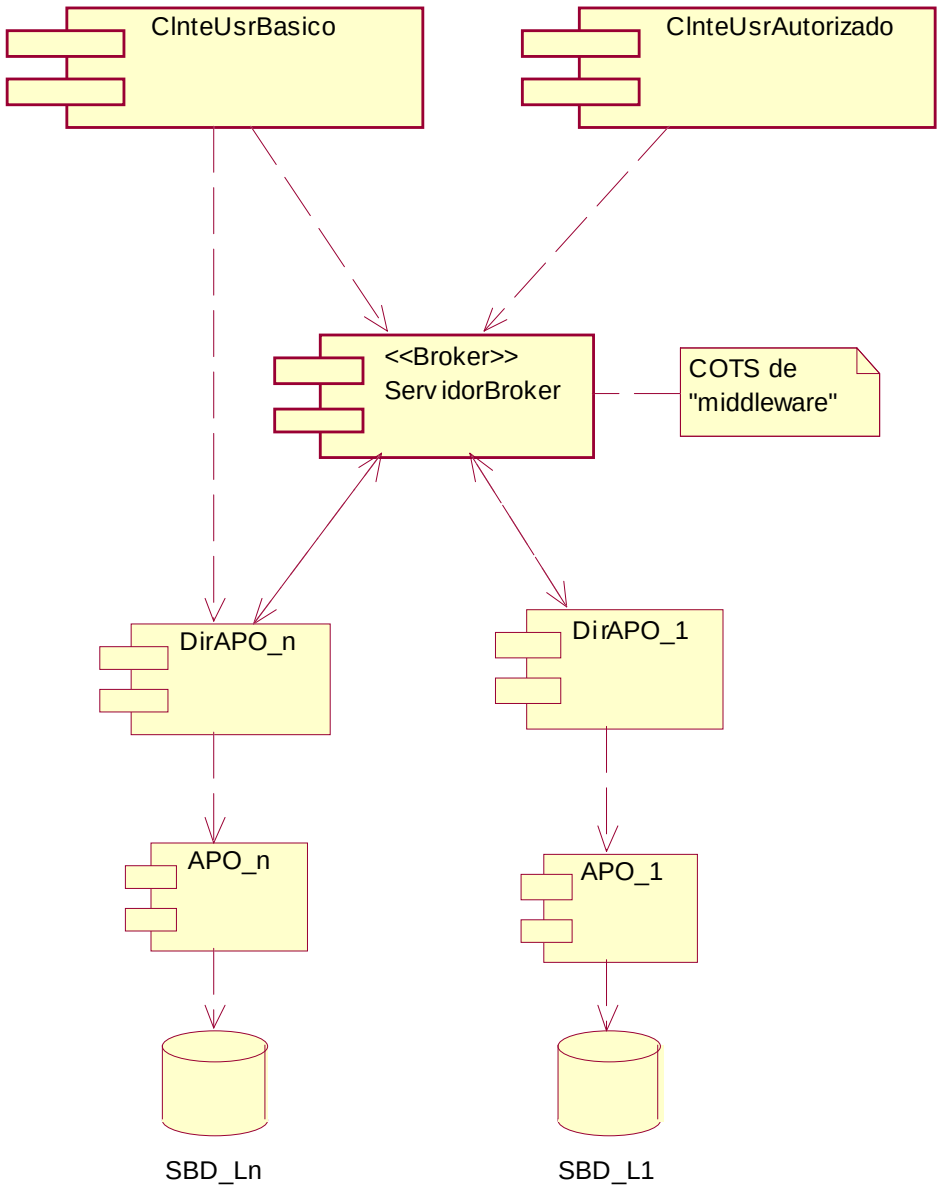

Figura #5: Aplicación de patrones de “Multicapas” y “Broker” de [POSA1] y [Gomaa 2004] en el diseño arquitectónico de DirAPO.

Entre los requerimientos no funcionales del sistema ilustrativo del apéndice, surge la necesidad de un ‘broker’ para desacoplar los servidores locales de las empresas parte de una corporación entre sí y con los clientes. En este requerimiento se alude precisamente a la característica de “transparencia de localización” que explica Gomaa en la cita anterior. La idea es que los servidores del sistema que requiera instalar cada una de las empresas de una corporación registran sus servicios ante el ‘broker’. Luego, tanto los usuarios que son

empleados de alguna empresa de la corporación como los que pertenecen a otras empresas (probablemente clientes o proveedores de alguna empresa de la corporación) tendrán acceso a los servicios a través del 'broker'. Si la localización del servidor de una de las empresas de la corporación cambia, ninguno de los clientes del 'broker' se verá afectado por el cambio, solo el 'broker' deberá ser notificado. Si eventualmente otra empresa se une a la corporación, inscribe su servidor ante el 'broker' y quedará accesible inmediatamente para todos los usuarios.

En resumen, el diseño arquitectónico es considerado por muchos autores una actividad compleja y decisiva (véase por ejemplo capítulo #6 de [Bruegge 2002]). Hay coincidencia en que el diseño arquitectónico requiere de ingenieros experimentados. Aunque la enorme difusión de "Multicapas" nos permite verlo casi como obvio en la actualidad, tomó muchos años de desarrollo tecnológico y metodológico concretar esta idea. Aún conociendo el patrón, la definición de la función principal de cada capa, dónde comienzan y terminan las responsabilidades de una capa, y cómo deben apoyarse en otras, no es un problema que se pueda dejar en manos de ingenieros novatos cuando se trata de sistemas complejos (véase pág. 50 de [POSA1]). Por tanto, el uso de patrones arquitectónicos puede mitigar riesgos importantes derivados de la falta de experiencia en el equipo de desarrolladores.

3.4 Características y beneficios del uso de patrones de diseño

Entre los patrones orientados a facilitar las actividades del proceso de construcción de software, los patrones de diseño adquieren la connotación de prototípicos. El catálogo [GoF] es un hito para la comunidad de ingenieros de software que estudian, usan y producen patrones. Con respecto de la naturaleza particular de estos patrones es poco lo que se puede agregar, solamente que están enfocados a esa etapa del proceso de construcción de software donde se llega después de haber trabajado el análisis y modelado los requerimientos así como el diseño arquitectónico, y constituye a su vez el preámbulo para un diseño detallado que culmina con la programación. "Un patrón de diseño nombra, abstrae e identifica aspectos clave de una estructura de diseño frecuente y común, lo que lo hace útil para crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases participantes y las instancias, sus funciones y colaboraciones, así como la distribución de responsabilidades. Cada patrón de diseño enfoca un problema o tema de diseño orientado a objetos particular. Describe cuándo se puede aplicar, si puede aplicarse en el contexto de ciertas restricciones de diseño y las consecuencias y aspectos alternativos derivados de su uso." (pág. 4 de [GoF]).

La figura #6 intenta dar una vista panorámica y localizada de los distintos tipos de patrones asociados con el proceso de desarrollo. Como se puede ver, los patrones de diseño se ubican en medio de la programación y el análisis y "antes de" los patrones arquitectónicos. También se puede apreciar que estos patrones son bastante independientes del dominio de aplicación, cosa que claro está no sucede con los patrones de análisis y se da "a medias" con los patrones arquitectónicos. Este diagrama también muestra otros tipos de patrones asociados con el proceso de desarrollo que no se han mencionado directamente en este informe y de los cuales se han publicado algunos catálogos.

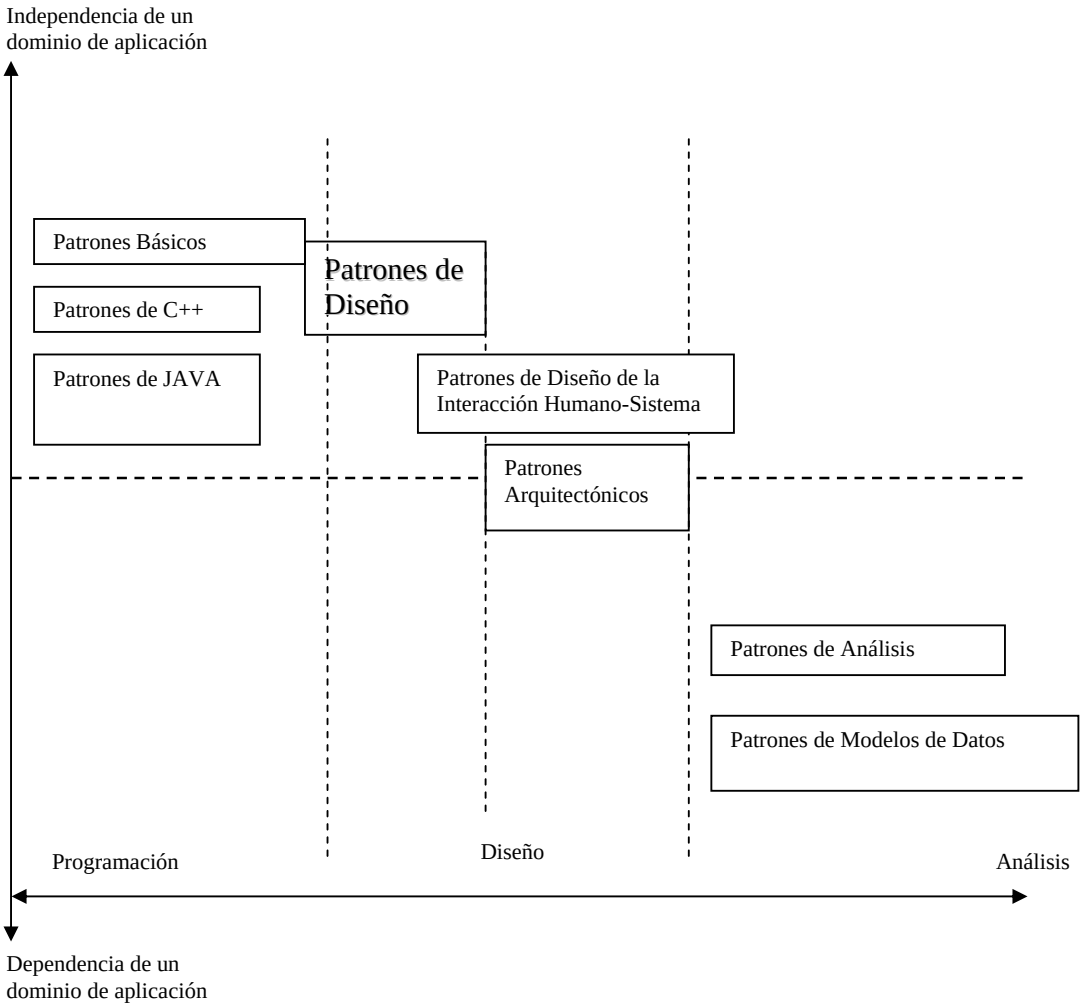

Figura #6: Ubicación de patrones de diseño

En el caso de estudio del apéndice se han usado patrones de diseño como “Composición”, “Cadena de Responsabilidades”, “Intérprete”, “Fachada” y “Proxy” de [GoF]. A continuación se explica brevemente el patrón “Composición” y una de sus aplicaciones en el contexto del problema que se analiza en el Apéndice 1.

“Composición” es un patrón muy apropiado para introducir el concepto de patrón de diseño tal como se plantea en [GoF]. La intención, según este catálogo, es: “Componer objetos en estructuras de árbol para representar jerarquías totalidad-parte. ‘Composición’ permite a los clientes trabajar uniformemente con los objetos individuales y con las composiciones de objetos.” (pág. 163 de [GoF]). La estructura general de la solución se muestra en la figura #7.

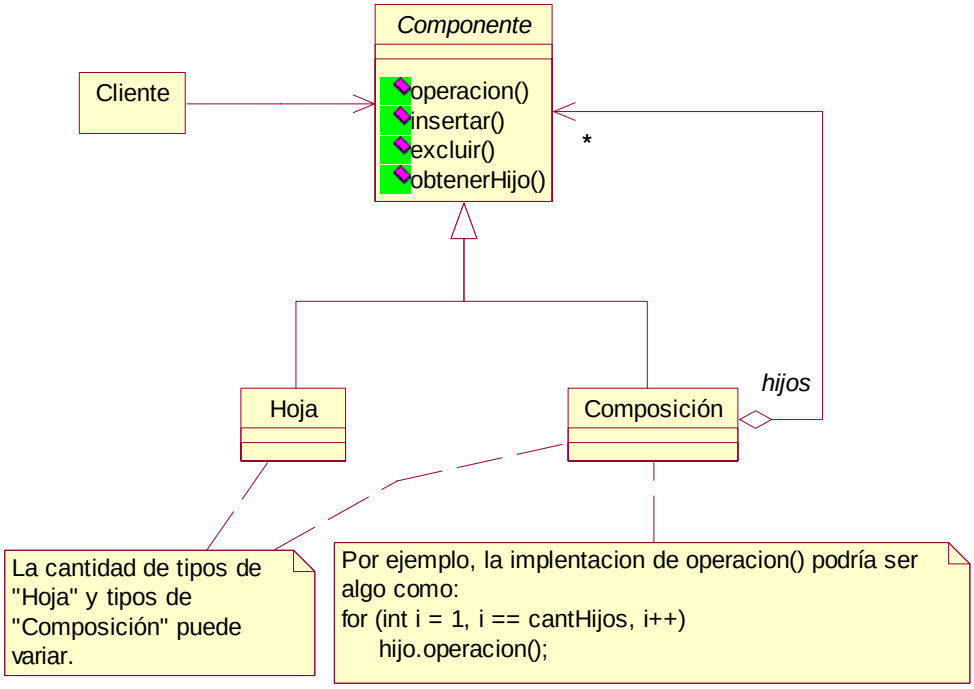

Figura #7: Estructura de solución de “Composición” de [GoF].

En la gran mayoría de los sistemas de información empresarial surge en algún momento la necesidad de representar clasificaciones de entidades u objetos. La clasificación de cuentas en un sistema de contabilidad, la clasificación de tipos de producto en un sistema de inventarios, la clasificación de puestos en un sistema de planillas son tan sólo tres buenos ejemplos de cómo se reitera la jerarquía de clases en el dominio de las aplicaciones empresariales. Es posible entonces pensar en un componente reutilizable que, bien diseñado y programado, pueda ser reutilizado en todos estos casos y otros similares. La idea es que cada vez que un subsistema requiera los servicios de un hipotético componente “ClasificadorGenerico”, éste se instanciaría con los datos de las clases específicas que el subsistema cliente requiere representar. El componente clasificador tendría que obtener de la base de datos los datos de las clases de la jerarquía que se requiere manejar, por lo que tendría su propio objeto para conectarse a la base de datos. Debido a que este componente podría recibir una solicitud de un subsistema remoto, sería conveniente incluir objetos de valor para administrar eficientemente el traslado de datos entre el clasificador y su cliente. En la figura #8 se muestra el diseño de clases de análisis de este componente. Claramente puede apreciarse el uso de “Composición”.

Esta aplicación del patrón no solo permite instanciarlo dinámicamente con jerarquías completamente distintas en cuanto a cantidad de niveles y cantidad de clases, sino que también es fácil extender el diseño para que se puedan eventualmente manejar distintos tipos de clases compuestas y de clases simples. En tal caso se pueden agregar clases derivadas de “ClaseHoja” y “ClaseCmp”.

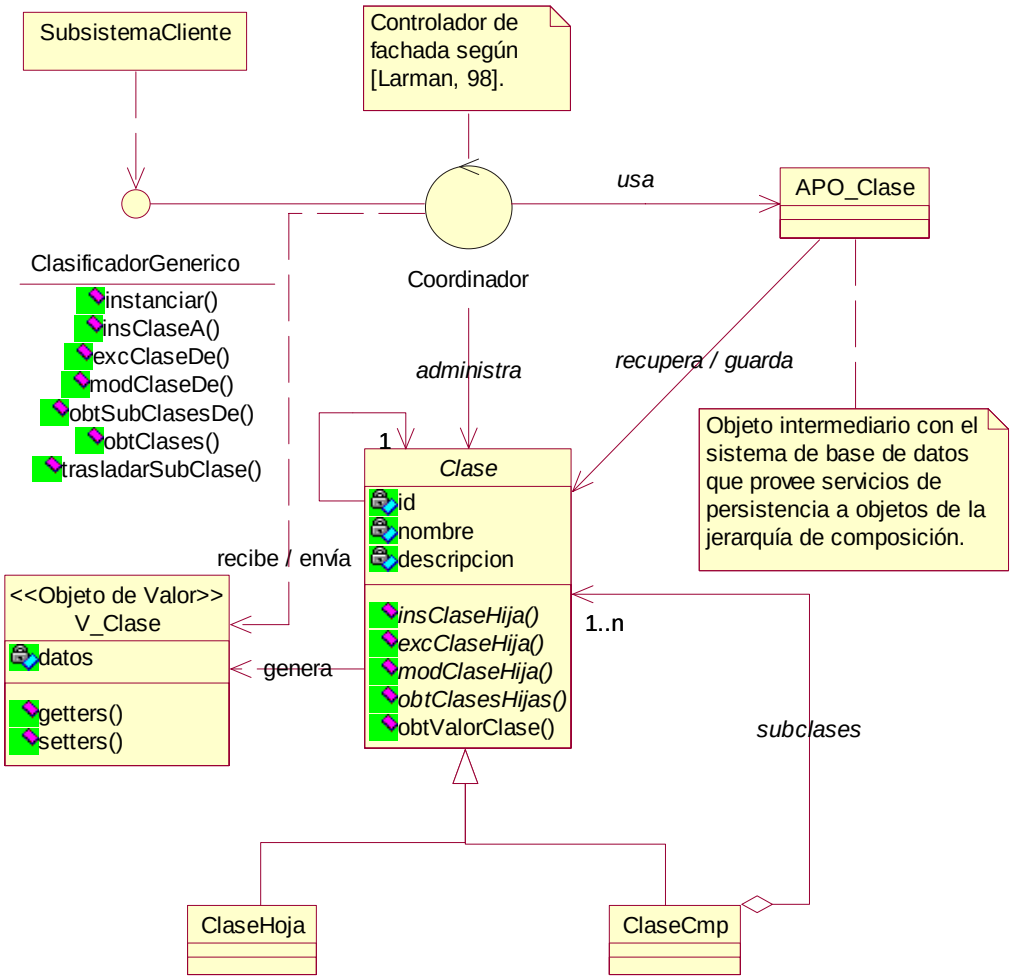

Figura #8: Aplicación de "Composición" de [GoF] para representar jerarquías de clasificación en general, encapsuladas en un componente especializado.

"Composición" se usa cuando se quiere dar el mismo tratamiento, en el nivel de programación, tanto a los objetos simples como a los compuestos. En este caso, la programación de la graficación de una jerarquía se puede facilitar si "ClaseHoja" y "ClaseCmp" responden de manera equivalente a una operación "graficar()", debido a que la interfaz genérica se especifica en la clase abstracta "Clase".

En resumen, en [GoF] se añade que los patrones de diseño contribuyen a resolver muchos problemas cotidianos de diseño que los ingenieros de software deben enfrentar, tales como:

1. Identificar los objetos apropiados: "Los patrones de diseño le permiten a usted identificar abstracciones menos obvias y los objetos que podrían representarlas. Por ejemplo, objetos que representan un proceso o algoritmo no aparecen típicamente en la naturaleza, sin embargo son cruciales en la flexibilidad de un diseño" (pág. 13 de [GoF]).

2. Determinar la granularidad apropiada de los objetos: “Los objetos pueden variar tremendamente en tamaño y cantidad. Pueden representar todo, desde componentes de bajo nivel del hardware, subiendo hasta lo más alto para representar aplicaciones completas. ¿Cómo decidimos qué puede ser un objeto? ... Los patrones de diseño pueden tratar este tema también.” (pág. 13 de [GoF]).
3. Especificar las interfaces de los objetos: “Los patrones de diseño le ayudan a usted a definir interfaces por medio de la identificación de sus aspectos clave y los tipos de datos que se pasan a través de una interfaz. Un patrón de diseño también puede indicarle a usted qué **no** debe poner en una interfaz....Los patrones de diseño también especifican relaciones entre interfaces...” (pág. 14 de [GoF]).
4. Aclarar detalles de la programación de los objetos. Este catálogo incluye numerosos consejos sobre cómo programar los patrones que sustentan buenas prácticas de programación orientada a objetos: ¿cómo usar la técnica de herencia?, ¿cuándo es mejor usar delegación por composición en lugar de herencia?, ¿por qué se debe programar apoyándose en interfaces en lugar de implementaciones que podrían estar sujetas a cambios y otros detalles similares (véase páginas 14 a 18 de [GoF]).
5. Poner a funcionar de manera efectiva los mecanismos de reutilización. Este es uno de los temas centrales en casi todos los patrones, no solo cómo el esquema genérico de solución provisto puede ser reutilizado, sino también cómo un componente diseñado con base en estos patrones resulta ser mucho más reutilizable (págs. 18 a 22 de [GoF]).
6. Relacionar de manera clara lo que sucede en tiempo de ejecución con las estructuras estáticas de tiempo de compilación: “La estructura de un programa orientado a objetos en tiempo de ejecución con frecuencia se parece muy poco a la estructura de su código. La estructura del código está congelada en tiempo de compilación, consiste de clases relacionadas estáticamente por herencia. La estructura de un programa en tiempo de ejecución consiste de redes rápidamente cambiantes de objetos que se intercomunican. De hecho ambas estructuras son bastante independientes. Intentar entender una a partir de la otra es como tratar de entender la dinámica de un ecosistema vivo a partir de la estructura taxonómica de las plantas y animales que lo habitan, y viceversa...Muchos patrones de diseño (en particular aquéllos que representan la dinámica de instancias u objetos) representan explícitamente la distinción entre estructura de tiempo real y estructura de compilación” (véase págs. 22 y 23 de [GoF]).
7. Diseñar con disposición al cambio. En relación con este aspecto, los autores de [GoF] señalan cómo algunos de los patrones del catálogo permiten mitigar factores frecuentes que implican rediseño en los sistemas, tales como:
    a. Creación de objetos mediante la especificación explícita de una clase.
    b. Dependencia de operaciones específicas.
    c. Dependencia de la plataforma de hardware o software.
    d. Dependencia de la representación o implantación de un objeto.
    e. Dependencias de algoritmos.
    f. Acoplamiento fuerte.
    g. Extensión de funciones mediante subclases.
    h. Imposibilidad de alterar clases convenientemente.

4. Patrones y sistematización del conocimiento

El concepto de “lenguaje de patrones” propuesto por Alexander ha sido, en nuestra opinión, poco comprendido en la comunidad de ingenieros de software. Se ha reducido el concepto a un esquema de utilidad técnica, una herramienta para el desarrollador de software en su quehacer cotidiano y se ha perdido la dimensión de lenguaje que Alexander le ha atribuido originalmente. Es por esto que en la investigación [Calderón 2003] se ha intentado rescatar la propuesta original reinterpretándola en función de las necesidades de sistematización del conocimiento en la industria del software. En esta sección se resume y contextualiza ese trabajo al ámbito del desarrollo de “familias de productos de software”. Las siguientes citas provienen todas de [Alexander 1979]:

1. "La gente puede darle forma a edificios, y lo ha hecho por siglos, usando lo que yo llamo lenguajes de patrones. Un lenguaje de patrones provee a cada persona que lo usa el poder de crear una infinita variedad de edificios únicos y nuevos, de la misma forma en que su lenguaje ordinario le da el poder de crear una variedad infinita de oraciones." (página 167).
2. "Es en este sentido que un sistema de patrones forma un lenguaje. Cuando el constructor de establos aplica los patrones de un establo, uno a otro en un orden apropiado, es capaz de crear un establo. El establo siempre mantendrá las relaciones requeridas por los patrones; sin embargo, los tamaños, los ángulos, y las relaciones dependen de las necesidades de la situación y de la inspiración del constructor. La familia de establos producida por este sistema comparte todas las características morfológicas especificadas en las reglas (éstas son las leyes morfológicas que hemos indicado), pero más allá de esto, existe una variedad literalmente infinita." (página 183).
3. "Un lenguaje de patrones es un sistema más complejo de esta categoría[8]. Los elementos son patrones. Los patrones tienen estructura, que muestra cómo un patrón es en sí mismo un patrón de patrones más pequeños[9]. Y también hay reglas, incluidas en los patrones, las cuales describen la forma en que pueden crearse, y la forma en que deben combinarse respecto de otros patrones. Sin embargo, en este caso, los patrones son a la vez elementos y reglas, de tal manera que las reglas y los elementos son indistinguibles. Los patrones son elementos. Y cada patrón es también una regla que describe las posibles combinaciones de los elementos—que a su vez son patrones." (página 185).
4. "Un lenguaje ordinario como el Inglés es un sistema que nos permite crear una variedad infinita de combinaciones unidimensionales de palabras, llamadas oraciones." (página 185).
5. "Un lenguaje de patrones es un sistema que nos permite crear una variedad infinita de combinaciones tridimensionales de patrones que llamamos edificios, jardines, pueblos." (página 186).
6. "En resumen: ambos, tanto los lenguajes ordinarios como los lenguajes de patrones, son sistemas combinatorios finitos que nos permiten crear a voluntad una variedad infinita de combinaciones únicas, apropiadas a circunstancias diferentes.

[8] Aquí el autor se refiere a una categoría de sistemas como la de los lenguajes naturales.
[9] Menos literal, podría ser la traducción "más detallados".

Lenguajes naturales	**Languajes de patrones**
Palabras	Patrones
Reglas gramaticales y significados que originan asociaciones	Patrones que especifican asociaciones entre patrones
Oraciones	Edificios y lugares

7. "A este punto, hemos definido el concepto de lenguaje de patrones claramente. Sabemos que es un sistema finito de reglas que una persona puede usar para generar una variedad infinita de edificios diferentes—todos miembros de una familia—y que el uso del lenguaje permitirá a la gente de un pueblo o ciudad generar el balance exacto entre uniformidad y variedad que da vida a un lugar. " (página 191).

Parafraseando a Alexander: "Un lenguaje de patrones provee a cada *ingeniero de software* que lo usa el poder de crear una infinita variedad de *sistemas de software* únicos y nuevos, de la misma forma en que su lenguaje ordinario le da el poder de crear una variedad infinita de oraciones" (véase la cita anterior de la página 67 de [Alexander 1979]). Desde la perspectiva de que el lenguaje natural *es* conocimiento, lo que Alexander propone en el fondo es que los lenguajes de patrones son sistematizaciones del conocimiento especializado de los arquitectos, en nuestro caso de los ingenieros de software. Ese poder creativo que Alexander asocia al lenguaje natural y en particular a los lenguajes de patrones, se hace más significativo en el contexto del desarrollo de software orientado a "familias de productos de software (FPS)" o "software product lines (SPL)". "Una SPL es un conjunto de sistemas intensivos en software que comparten un conjunto administrado de características que satisfacen las necesidades específicas de un segmento particular de un mercado o de un tipo de misión crítica y que son construidos a partir de un conjunto básico de activos de una manera prescrita." (pág. 5 de [Clements 2002]). Un ejemplo típico de una FPS es el conjunto de aplicaciones para ofimática de Microsoft que incluye MS-Word, MS-Excel, MS-Power Point, etc. Como usuarios hemos podido experimentar la evolución de esta familia a lo largo del tiempo y la mejora continua en la integración de las aplicaciones de la familia. En este caso específico se puede decir que la familia está constituida por varios tipos de productos. En otros casos, una FPS consiste básicamente en un tipo de producto que evoluciona gradualmente o se personaliza fácilmente para adecuarlo a las necesidades de clientes específicos.

El concepto de FPS está fuertemente conectado con el de "dominio" o más específicamente "dominio de aplicación": "Un dominio es un cuerpo de conocimientos especializados, un área de pericia[10], o una colección de funcionalidad relacionada. Por ejemplo, el dominio de telecomunicaciones es un conjunto de funcionalidades de telecomunicación, que a su vez consiste de otros dominios tales como la conmutación (el "switching"), los protocolos, la telefonía y las redes. Una línea de productos de software es un conjunto específico de sistemas de software que provee parte de esta funcionalidad" (pág. 14 de [Clements 2002]). Entonces re-parafraseando: "Un lenguaje de patrones orientado a un dominio de aplicación específico provee a cada *ingeniero de software* que lo usa el poder de crear una infinita variedad de *productos de software* únicos y nuevos, pertenecientes a una misma FPS (o SPL), de la misma forma en que su lenguaje ordinario le da el poder de crear una variedad infinita de oraciones". Creo que esta es una mejor manera de empezar a transferir

[10] Se usa "pericia" por "expertise" en Inglés.

efectivamente y pragmáticamente el concepto de *lenguaje de patrones*, propuesto por Alexander en la arquitectura, al contexto de la ingeniería del software.

Por otro lado, si se acepta que el conocimiento en general es fundamental para el éxito de una empresa y que por ende se debe sistematizar y potenciar su uso, entonces cabe preguntarse ¿qué significa esto para una empresa desarrolladora de software? ¿Cuáles son las distintas categorías de conocimiento que una empresa desarrolladora de software debería sistematizar y potenciar? ¿Cómo puede hacerlo? Claramente son muchas las categorías de conocimiento que una empresa desarrolladora de software debería sistematizar y potenciar: conocimiento sobre el mercado en que compite, conocimiento sobre métodos de desarrollo de software que ha usado, conocimiento sobre estándares y técnicas específicas, conocimiento sobre tecnologías de desarrollo, conocimiento sobre la administración del proceso de desarrollo, etc. Sin embargo hay una categoría que es particularmente costosa y decisiva, se trata del conocimiento que año a año, proyecto a proyecto, sus analistas y programadores construyen en relación con un dominio de aplicación o de negocio específico, es decir como resultado de sus experiencias en un dominio de aplicación específico. Por ejemplo el conocimiento especializado que construyen los analistas y programadores de una empresa que se ha especializado en aplicaciones de administración hospitalaria, o el conocimiento asociado a sistemas de información empresarial para la industria manufacturera, etc. Esta categoría de conocimiento tiene otra particularidad y es que pertenece menos al dominio de la ingeniería del software (como disciplina especializada en la producción de software) y más al dominio del negocio hacia donde se orientan los productos de software de una empresa. Este categoría de conocimiento es un puente entre el personal técnico de un proyecto de desarrollo y los usuarios, clientes o expertos del dominio de los negocios meta. En este sentido cabe destacar que, con los lenguajes de patrones, Alexander buscaba facilitar la participación en el diseño de sus espacios arquitectónicos a las personas que lo iban a habitar (véase la cita número siete en la página anterior). Esta a su vez ha sido una búsqueda constante y cada vez más apreciada por los ingenieros de software[11].

Consecuentemente, la perspectiva con que se tratan los lenguajes de patrones en este informe busca enfocar dos necesidades imperiosas de las empresas productoras de software:

1. Preservar el conocimiento experto que adquieren y perfeccionan sus desarrolladores de software (arquitectos, analistas y programadores) en dominios específicos (informática hospitalaria, informática jurídica, sistemas de información empresarial, etc.) ante la imposibilidad muchas veces de retener al personal que ha ido poco a poco, durante varios proyectos y años, adquiriendo una pericia invaluable, acumulando un capital de conocimientos, en un dominio dado.
2. Sacar el máximo provecho a este capital de conocimientos. Esto implica que la empresa debe:
    a. Aplicar efectiva y eficientemente el conocimiento acumulado durante el proceso de desarrollo de software. Lo que lleva a:
        i. Que los ingenieros de software dispongan de los mecanismos que faciliten la aplicación de un lenguaje de patrones a la construcción de un producto de software perteneciente a una FPS.

[11] Considérese por ejemplo los distintos puntos de verificación en el modelo de proceso de espiral de Bohem, o el trabajo directo con los usuarios y clientes en los procesos de "prototipaje" e inclusive el tipo de involucramiento de usuarios y clientes en métodos ágiles como "Extreme Programming".

ii. Que los ingenieros de software y los clientes, usuarios o expertos en el dominio del negocio, dispongan de un lenguaje mutuamente inteligible y accesible para garantizar en el desarrollo del producto la participación plena y efectiva de clientes, usuarios o expertos en el dominio. Esto con el fin de reducir los riesgos asociados al análisis de requerimientos. Después de todo, este era uno de los objetivos fundamentales para Alexander en la arquitectura y ha sido una búsqueda incesante para los ingenieros de software.

b. Difundirlo oportunamente entre los distintos equipos de trabajo y los ingenieros de software de tal forma que su aprendizaje sea fácil para los desarrolladores novatos en el dominio.
c. Integrarlo, contrastarlo y enriquecerlo con el conocimiento de patrones de análisis disponible a través de publicaciones como las memorias de las conferencias PLoP u otros catálogos (por ejemplo [Fowler 1997], [Coad 1992], [Coad 1997], Hay [Hay 1996], [Silverston 1996] y [Ericksson 2000] en el dominio de los sistemas de información empresarial).
d. Integrarlo efectiva y eficientemente con el conocimiento de patrones arquitectónicos y patrones de diseño.
e. Administrarlo apropiadamente.

En síntesis, en este informe se busca profundizar en el concepto de "lenguaje de patrones" con la intención de mejorar la competitividad de las empresas desarrolladoras de software, a través de un pilar fundamental como es el capital de conocimiento acumulado en el contexto de un modelo de proceso cada vez más orientado al desarrollo de familias de productos de software en lugar de "productos aislados". Para lograr esto, la siguiente sección presenta, con un ejemplo, una forma de organizar el conocimiento de patrones que básicamente consiste en una síntesis de las propuestas de [POSA1], [POSA2] y otros catálogos como [GoF], [Alur 2001], [Fowler 1997], [Hay 1996] y [Silverston 1996], la cual se ha publicado en [Calderón 2003].

4.1 Principios para la representación de lenguajes de patrones orientados a dominios

Un catálogo de documentos electrónicos de patrones con facilidades de búsqueda sobre los documentos que especifican los patrones no sería suficiente para representar un lenguaje de patrones en general y por ende para organizar apropiadamente el conocimiento de patrones que una empresa ha construido en relación con un dominio de aplicación específico, es decir, un *lenguaje de patrones orientado a un dominio* o *un lenguaje de patrones de análisis de dominio*. En [POSA 1] se propone organizar y representar los lenguajes de patrones por medio de "sistemas de patrones": "Un sistema de patrones liga los patrones que lo constituyen. Describe cómo se conectan los patrones y cómo se complementan entre sí. Un sistema de patrones también da soporte efectivo al uso de patrones en el desarrollo de software" (pág. 360 de [POSA 1]). Luego agregan que "Un sistema de patrones para el diseño arquitectónico de software es una colección de patrones de diseño arquitectónico, junto con los lineamientos para su aplicación en la programación, su combinación y uso práctico en el desarrollo de software". Para especificar aún más esta definición escueta, indican los requerimientos de un sistema de patrones:

- "Debe abarcar una base suficiente de patrones...

- Debe describir todos sus patrones uniformemente...
- Debe evidenciar las distintas conexiones entre los patrones...
- Debe organizar los patrones que lo constituyen...
- Debe soportar la construcción de sistemas de software...
- Debe soportar su propia evolución...” (págs. 361-362 de [POSA1]).

En [POSA2] se presenta una organización alternativa que busca destacar fuertemente las conexiones entre los patrones por medio de un diagrama de la red de patrones y que, a juicio de este nuevo grupo de autores (que solo incluye a dos de los coautores de [POSA1] de un total de cinco), es mejor que la representación lograda en [POSA1]. En [POSA2] se hace mayor énfasis en la organización del conocimiento de lenguajes de patrones. Desde la perspectiva del desarrollo de FPS, esta nueva propuesta sugiere:

1. Facilitar la identificación de áreas de problemas de diseño en una FPS, identificar los patrones que resuelven estas áreas y sus interconexiones relevantes desde el punto de vista de la construcción de productos de la FPS.
2. Facilitar la identificación de conexiones entre las áreas de problemas de diseño de una FPS.
3. Facilitar la identificación de patrones arquitectónicos como “puntos de entrada” a una red de patrones que orienta el diseño de productos de una FPS.
4. Destacar las conexiones del tipo “usa” o “complementa” entre patrones, que revelan cómo la aplicación de un patrón puede complementarse con la aplicación de otro para resolver problemas reales de diseño en el contexto de una FPS.

En [Calderón 2003] se ha planteado una síntesis de las propuestas de [POSA1] y [POSA2] incluyendo características valiosas de los esquemas organizativos de otros catálogos como [GoF], [Alur 2001], [Fowler 1997], [Hay 1996] y [Silverston 1996]. Para diferenciar esta propuesta de otras se ha usado el término “lexicón de patrones”. En términos generales un “lexicón de patrones” permite representar un lenguaje de patrones, particularmente un lenguaje de patrones de análisis para un dominio específico, de tal manera que organiza el conocimiento de patrones de análisis en un dominio específico de forma natural para facilitar su aprendizaje y evolución. En el concepto de “lexicón de patrones” se ha querido enfatizar la organización del conocimiento con el propósito de facilitar el aprendizaje por lo que se han propuesto los siguientes principios organizativos:

1. Dado un dominio de aplicación, los patrones se organizarán en tres niveles de categorización: nivel supraordinal (constituido por categorías de patrones e inclusive dominios afines), nivel básico o común que incluirá a los patrones propiamente y nivel subordinal que incluirá a las variantes y combinaciones de patrones del dominio.
2. No solo patrones, sino también categorías de patrones y dominios afines son consustanciales a un lenguaje de patrones de análisis para un dominio específico, por tanto deberán ser representados en un “lexicón de patrones” orientado a un dominio de aplicación específico. Las categorías de patrones están asociadas con áreas de problemas de diseño de una FPS. En los distintos catálogos referidos se pueden encontrar categorías de patrones tales como “Patrones de construcción”, “Patrones de Comportamiento” y “Patrones de Estructura” en [GoF]; “From Mud to Structure” y “Management” de [POSA 1]. En [POSA 1] se identifican tres grandes dominios de patrones: “Patrones de diseño arquitectónico”, “Patrones de diseño” e “Idioms” o patrones de diseño detallado de objetos, asociados a un lenguaje de programación específico.

3. Igualmente son consustanciales a los lenguajes de patrones las categorías de conexiones o relaciones entre patrones. Algunos ejemplos de categorías de patrones que aparecen con frecuencia en catálogos de patrones (y que se pueden representar como estereotipos de UML para facilitar su representación gráfica) son <<soporta a>>, <<implementado con>>, <<usado con>>, <<contrasta con>>, <<similar a>>. Entre patrones de análisis de dominio aparecen conexiones relevantes como <<evoluciona a>> o <<se combinan en>> para representar combinaciones típicas de patrones.
4. Las conexiones entre categorías de patrones y entre dominios afines también deben ser considerados elementos fundamentales de un lenguaje de patrones y por tanto tendrán su representación adecuada en un "lexicón de patrones". Estas conexiones deberán siempre sustentarse en conexiones entre patrones incluidos en las categorías o dominios asociados.
5. A lo largo de la evolución de un lenguaje de patrones surgirán de manera natural combinaciones de patrones y variantes de patrones que se repiten con frecuencia, es decir que a su vez constituyen patrones. Por tanto un "lexicón de patrones" representará y organizará variantes y combinaciones típicas de patrones en el nivel subordinal de categorización.
6. Las categorías de patrones tienen una estructura compleja, no se trata simplemente de listas de patrones. En las categorías aparecen patrones centrales y otros relativamente periféricos, buenos y no tan buenos ejemplos de la categoría. Con frecuencia los autores de catálogos de patrones de análisis inician la exposición de una categoría de patrones con patrones muy simples que luego extienden paulatinamente. Estos patrones incluyen diferenciaciones que son fundamentales en el dominio y por tanto pueden considerarse buenos ejemplos de la categoría o dominio de patrones.
7. Los patrones centrales en una categoría de patrones se denominan "patrones prototípicos". Un patrón prototípico siempre será más fácil de aprender que sus comiembros de categoría. Al ser más simple que sus comiembros de categoría, un patrón prototípico deberá sugerir una imagen o metáfora que pueda ser usada para representar no solamente su propia intencionalidad sino también la categoría como un todo. Por tanto, identificar patrones prototípicos es muy útil para compartir conocimiento del dominio en la empresa desarrolladora de software y muy especialmente con desarrolladores que tienen poca experiencia en el dominio. Un patrón prototípico es un buen punto de inicio para una búsqueda de patrones si se complementa con conexiones del tipo <<similar a>> y <<contrasta con>> para facilitar la búsqueda de patrones apropiados a un contexto de análisis específico. En el mismo sentido, la identificación clara de "categorías centrales de patrones" en el nivel supraordinal y la representación de conexiones con otras categorías, puede servir a los mismos fines.
8. Por supuesto, es importante incluir en un lexicón de patrones de análisis de dominio la identificación explícita de patrones que son "puntos de entrada" y las cadenas de patrones que se pueden seguir para completar un análisis. Estos puntos de entrada a cadenas de patrones son muy apreciadas por los ingenieros de software pues definen grandes rutas del análisis, por esto mismo han recibido especial atención por parte de los autores de catálogos de patrones. En un lexicón de patrones de análisis de dominio los "puntos de entrada" serán patrones arquitectónicos relevantes para una FPS.
9. En un lexicón de patrones de análisis de dominio, un patrón puede aparecer como miembro de varias categorías, esto no solo refleja las fronteras difusas entre los dominios relevantes para una FPS sino que también facilita distintas formas de acceder un mismo patrón, dependiendo del análisis que un desarrollador esté llevando a cabo en un caso específico.

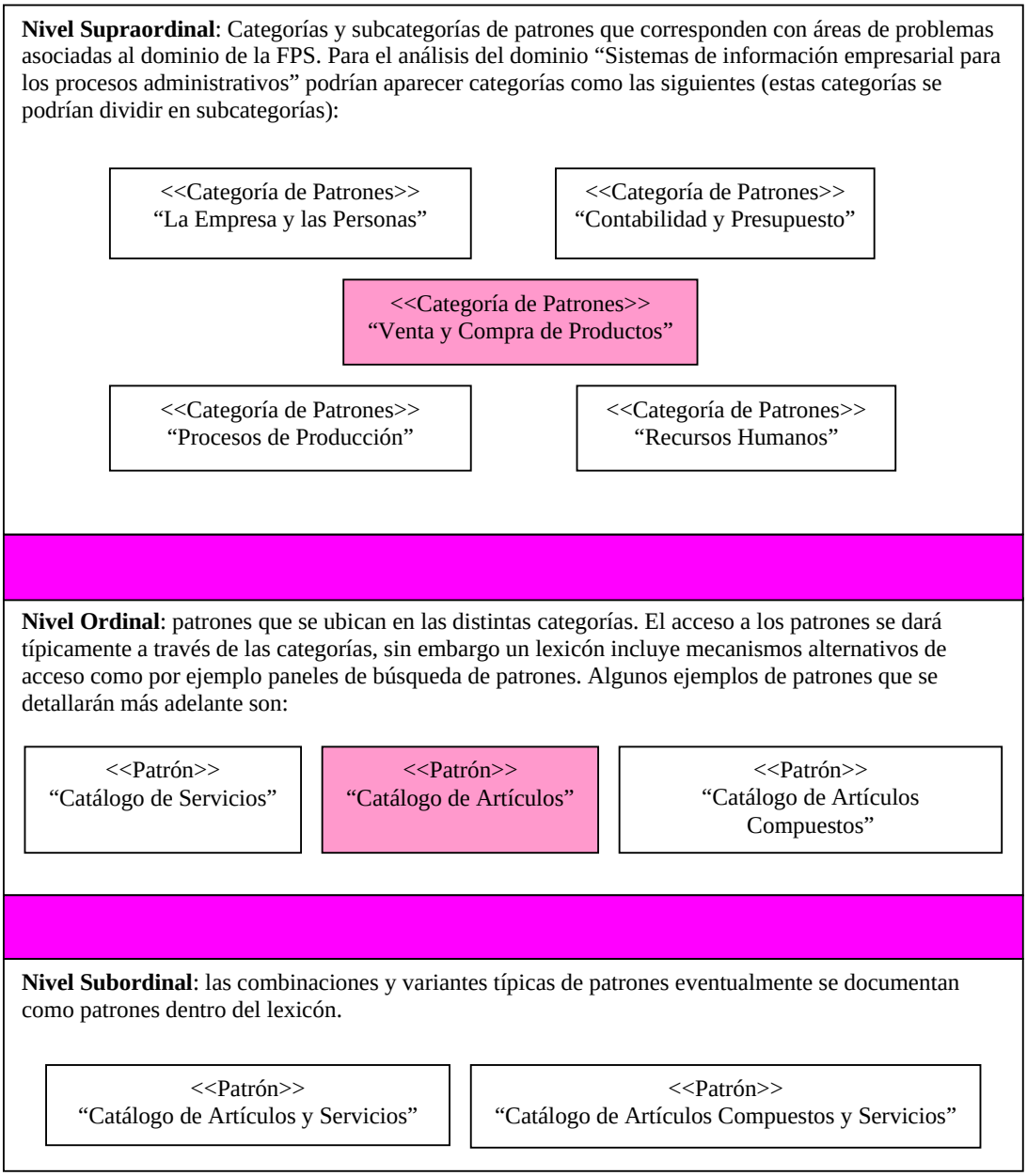

Figura #9: Estructura de categorización de un lexicón para el análisis del dominio "Sistemas de información empresarial para procesos administrativos".

4.2 Ejemplo de representación de un lenguaje de patrones para un dominio específico

En el siguiente ejemplo esquemático se muestra cómo se podrían organizar algunos de los patrones orientados a una FPS para el dominio de aplicación referenciado en la figura

anterior. La fuente principal para la especificación de estos patrones y categorías de patrones han sido los catálogos de [Hay 1996] y [Silverston 1996] que se centran en patrones de modelos de datos. Dado que evidentemente la mejor forma de representar un lexicón es a través de un hipertexto, aquí solamente se tratará de mostrar cómo se podrían organizar los distintos niveles del lexicón y algunos de los tipos de conexiones entre los distintos elementos. Se ha escogido presentar una "vista transversal" del lexicón centrada en la categoría de patrones "Venta y Compra de Productos" que se ha considerado una categoría central del dominio. Se ilustrará esta vista transversal a través de figuras en las que tanto las categorías de patrones como los patrones mismos son representados por medio de paquetes de UML. En el nivel ordinal y subordinal aparecerán patrones de análisis para el dominio escogido que han sido especificados utilizando UML y textos complementarios.

En la figura #10 se muestran las categorías principales del dominio bajo análisis. En este caso se ha considerado que la categoría central del dominio es "Venta y Compra de Productos", el argumento para dicha escogencia es que el análisis de esta área del dominio es prioritaria puesto que está asociada con los procesos que motivan la existencia misma de la empresa. Dicho de otro modo, las diferenciaciones que se hacen en esta categoría de patrones son fundamentales para que los productos de la FPS se logren alinear con los objetivos del negocio. En la figura #11 se especifica esta categoría central de patrones.

En la figura #12 se muestran las subcategorías de la categoría central escogida. Se ha considerado la subcategoría "Productos" como central dado que los patrones de esta subcategoría abarcan diferenciaciones básicas para que se puedan comprender los procesos de venta y compra cuyos patrones de análisis estarían incluidos en las otras dos subcategorías. Seguidamente en la figura #13 se especifica la subcategoría central "Productos" escogida en esta vista transversal del lexicón.

En la figura #14 aparece el conjunto de características funcionales que son consideradas en los distintos patrones de la figura #15. Los patrones abarcarán las combinaciones más comunes de características funcionales. Estas combinaciones comunes que son adoptadas por cada patrón constituyen la sección de "fuerzas" que se incluye en la especificación de cada patrón. Respecto de la relación entre requerimientos funcionales y características funcionales, Gomaa plantea que "…El término *requerimiento* es usado para denominar las necesidades que una línea de productos de software, y por ende al menos uno de los miembros de la línea de productos, debe ser capaz de satisfacer. Una vez que la línea de productos de software ha incluido y especificado este requerimiento, el requerimiento se denomina una *característica* provista por la línea de productos de software" (pág. 95 de [Gomaa 2004]). El análisis de características comunes y su variabilidad en una FPS es nuevo para quienes no estén familiarizados con este enfoque, no obstante es considerada una actividad crucial por los desarrolladores de FPS. No se dispone del espacio para explicar el análisis de características, sin embargo, brevemente cabe señalar que se identifican características comunes a todos los productos y características opcionales. Las características se agrupan en subconjuntos excluyentes o no excluyentes. En los subconjuntos excluyentes se puede identificar una característica estándar o típica y otras menos típicas que se denominan alternativas. En los conjuntos no excluyentes igualmente se puede identificar una característica estándar o típica y otras menos típicas que se denominan opcionales. La inclusión de algunas características en un producto puede requerir o inducir la inclusión de otras, en cuyo caso se establece que son características mutuamente incluyentes.

El análisis de características pretende responder a la variabilidad de necesidades en el tiempo y en el espacio de posibles clientes. En general, una categoría de problema en el análisis de un dominio consiste en identificar una *categoría de requerimientos* que típicamente aparecen asociados en las necesidades de las organizaciones cliente, analizar su variabilidad en el espacio mismo de las organizaciones y su variabilidad en el tiempo, es decir cómo se anticipa que puedan evolucionar dichas necesidades en una organización cualquiera. Por tanto, una categoría o subcategoría de patrones incluye la *categoría de características* correspondientes que podrán incluir los productos de una FPS para el dominio en cuestión; y los patrones incluidos en una categoría o subcategoría de patrones responden a combinaciones típicas de estas características como sus "fuerzas motivadoras" y ofrecen modelos o artefactos de utilidad en el análisis destinado a la construcción de productos de la FPS. A su vez, los patrones básicos de una categoría se combinan en nuevos patrones para responder a necesidades típicas más sofisticadas y evolucionan en formas anticipadas por el análisis de dominio de acuerdo con la evolución típica de las necesidades en las organizaciones cliente. Esto da origen a conexiones relevantes entre los patrones de una categoría de patrones.

En la figura #15 se muestran los patrones de la subcategoría "Productos". Se han considerado como patrones prototípicos de esta categoría a "Catálogo de Artículos" y a "Catálogo de Servicios" por cuanto se ha supuesto que representan las dos situaciones más simples que se pueden encontrar en una empresa. Estos dos patrones se pueden combinar para satisfacer necesidades de empresas que abarcan tanto artículos como servicios dentro de sus ventas. Un "Catálogo de Artículos" (simples) puede fácilmente evolucionar hacia un "Catálogo de Artículos Compuestos". Por otro lado, la combinación de características más sofisticada que se considera típica en esta categoría de patrones consiste en una empresa que pueda requerir no solo soportar la venta de artículos simples y servicios, sino también la venta de artículos compuestos. De esta forma el nivel ordinal del lexicón (patrones básicos) se representa junto con el nivel subordinal (combinaciones de patrones y variaciones de patrones). Otras combinaciones de características no han sido consideradas típicas y por tanto no se han representado con patrones. Por otro lado, algunas características de la categoría no aparecen en ninguno de los patrones, sin embargo sí se han incluido en el análisis de características. De esta manera, a través de los patrones se construyen los artefactos (específicamente modelos) de utilidad inmediata en la construcción de productos de la FPS, lo cual permite inducir cuáles son los componentes de software que son de utilidad inmediata para la construcción de productos. A la vez, con el análisis más amplio de características se especifican los alcances de la FPS desde el punto de vista de posibles productos que se podrían construir en plazos más amplios. Una FPS evolucionará hacia la inclusión de nuevos patrones conforme nuevas combinaciones típicas de características se consideren necesarias. Como ejemplos, las categorías de patrones "Venta y Compra de Productos" y "Productos" son suficientes para mostrar el valor práctico de las categorías de patrones en el contexto del análisis de dominio.

En las figuras #16 hasta #20 se incluyen documentos para especificar el patrón "Catálogo de Artículos". Luego en las figuras #21 hasta #25 para especificar el patrón "Catálogo de Servicios". La intención principal es mostrar cómo se podrían documentar patrones de análisis de dominio. Tal como se ha indicado, la fuente principal para la elaboración de estos patrones han sido los patrones de modelos de datos provistos por [Hay 1996] y [Silverston 1996]. Sin embargo cabe enfatizar que estos autores se han limitado precisamente a modelos de datos. Otros autores de patrones de análisis incluyen modelos conceptuales de objetos, flujos de trabajo y diagramas de secuencia o de colaboración (véase por ejemplo [Fernández

1999b] y [Fernández 2000]). Aunque esquemáticamente, se sugiere de esta manera una plantilla para la especificación de patrones de análisis orientados a dominios de aplicación específicos o patrones de análisis de dominio. El uso de patrones de análisis en el contexto de FPS no ha sido la intención de los autores que han publicado patrones de este tipo. La sugerencia principal consiste entonces en incluir un análisis de fuerzas en términos de características comunes, variantes y opcionales, así como un modelo de casos de uso ampliado con los objetos de frontera que detallan gráficamente los atributos y operaciones disponibles en cada caso de uso. El objetivo es integrar el análisis de dominio orientado a características, tal como ha sido planteado en [Gomaa 2004] con la actividad de especificación de patrones de análisis de dominio, dado que la identificación de características y su variabilidad es considerada como una tarea fundamental en el proceso de desarrollo de software orientado a FPS.

"…Ningún patrón es una isla…" nos recuerda Alexander en [Alexander 1979]. Por esta razón las conexiones entre patrones reciben especial atención en la construcción de un lexicón como el propuesto. Por la misma razón las conexiones entre categorías de patrones son muy importantes en la construcción de un lexicón de patrones. Las conexiones se han representado por medio de relaciones estereotipadas en UML. Cada uno de estos estereotipos tiene un significado preciso en el contexto en que aparece. En el caso de <<se complementa con>> entre las categorías de patrones del nivel supraordinal del lexicón en la figura #10, este tipo de conexión pretende indicar al analista, por ejemplo, que los patrones de las categorías "La Empresa y las Personas" y "Venta y Compra de Productos" se complementan mutuamente. Esto significa que al analizar la administración de los productos que una empresa ofrece, con frecuencia surgirá la necesidad de analizar la administración de sus clientes y proveedores, razón por la cual el analista podrá complementar su análisis en relación con los productos mediante los patrones de la categoría "La Empresa y las Personas" que incluyen conceptos y diferenciaciones típicas en relación con los clientes, proveedores y socios de una empresa. En la figura #15 aparecen los patrones de la categoría "Productos" y las conexiones entre estos. Tal como se ha comentado, <<evoluciona a>> representa un tipo de conexión cuya intención es anticipar la forma en que evolucionan las necesidades de cada empresa en relación con la administración de sus productos, esto es, la variabilidad en el tiempo. Por otro lado, la variabilidad de necesidades en el espacio de empresas se logra representar en el lexicón mediante conexiones del tipo <<se combinan en>>.

Finalmente, el concepto de lexicón de patrones propone sistematizar el conocimiento de expertos desarrolladores de software no solo en un dominio de aplicaciones central sino también en dominios adyacentes. Para el dominio de "Sistemas de información empresarial para procesos administrativos" los dominios "Interfaces humano-sistema para sistemas de información empresarial" y "Arquitectura de sistemas de información empresarial" son dominios adyacentes muy relevantes que deberían representarse en el lexicón de patrones también. En el primer dominio aparecerían patrones de interfaz humano-sistema que representan esquemas de interacción típicos en los sistemas de información empresarial. Estos patrones podrían complementar muy bien los patrones de análisis de dominio descritos porque sugerirían cómo "dar forma" a la interacción de los casos de uso identificados en tales patrones de análisis. Por otro lado, en el dominio "Arquitectura de sistemas de información empresarial" se podrían incluir las distintas configuraciones típicas de un sistema de información empresarial para procesos administrativos. Cada configuración típica se documentaría mediante un patrón arquitectónico que serviría como "punto de entrada" en el diseño arquitectónico de un producto con características particulares para un cliente

específico. De esta manera, la estructura de solución de un patrón arquitectónico mostraría los subsistemas y las conexiones entre estos que garanticen a una empresa cliente ciertas prestaciones para el manejo de la información asociada con sus procesos administrativos. Cada patrón arquitectónico en cierta forma representaría una categoría de cliente típico por medio de los requerimientos funcionales que satisface la arquitectura asociada al patrón. El tema del diseño arquitectónico es central en el desarrollo de software orientado a FPS, según indican [Jacobson 1997], [Clements 2002] y [Gomaa 2004] y de ahí se deriva la importancia de representar este dominio en un lexicón de patrones de análisis para el dominio "Sistemas de información empresarial para procesos administrativos". Los patrones arquitectónicos incluirían conexiones con patrones y categorías de patrones del dominio central del cual se ha presentado una vista transversal en las figuras #10 a la #25. De esta manera, si previo al análisis se ha acordado una arquitectura de producto para un cliente específico, el análisis continuaría con base en los patrones de análisis conectados a dicha configuración dentro del lexicón. Esto puede reducir significativamente el tiempo y los riesgos del análisis.

Un lexicón de patrones como el que se ha esquematizado puede servir además para identificar los componentes de software reutilizables que mejor se adaptan a la hora de construir un producto para un cliente específico. Mediante las recomendaciones para la aplicación (al análisis) de cada patrón en conjunto con las sugerencias para la implantación de un patrón de análisis (uso de patrones de diseño, de patrones de programación y componentes de software específicos) se describirá "…cómo los productos son construidos a partir del núcleo de activos…" (pág. 34 de [Clements 2002]) que es el objetivo principal del "Plan de Producción".

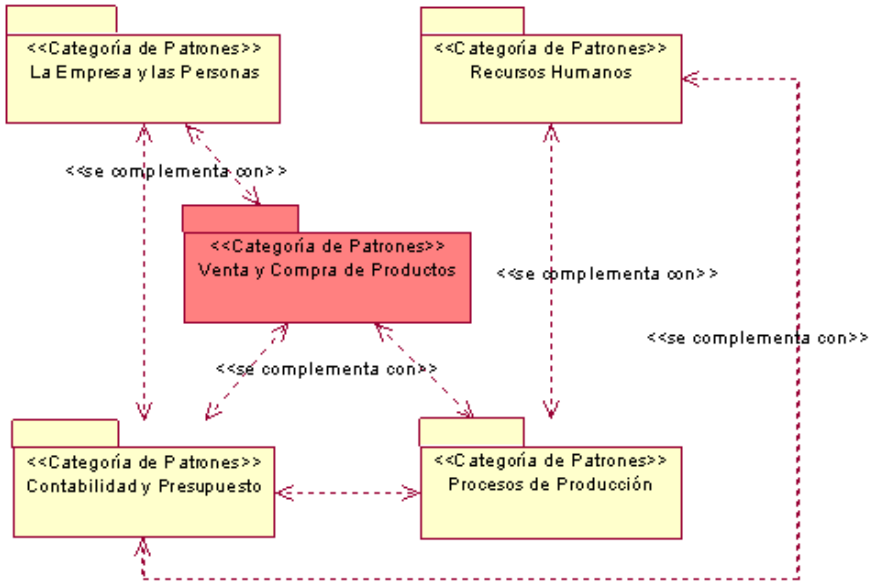

Figura #10: Nivel supraordinal del lexicón.

Intencionalidad: Los patrones de esta categoría generalizan artefactos del análisis que modelan requerimientos de información que tienen las organizaciones sobre sus productos y los procesos de venta y compra de dichos productos.

Algunos de estos requerimientos de información son, por ejemplo:

"1. ¿Cómo se comparan en calidad y precio los productos de la organización con los de la competencia?
2. ¿Qué nivel de existencias se requiere en cada localización para poder satisfacer las necesidades de los clientes?
3. ¿Cuál es el precio, el costo y la ganancia de los productos ofrecidos?
4. ¿Dónde pueden comprarse los mejores servicios y productos a los mejores precios?" (pág. 41 de [Silverston 1996]).

Terminología:
1. Producto: es un Producto Tangible o un Producto Intangible. Los productos intangibles se conocen típicamente como servicios.

2. Proceso de venta: proceso mediante el cual la organización provee productos o servicios a sus clientes obteniendo a cambio alguna retribución económica o de otro tipo.

3. Proceso de compra: proceso mediante el cual la organización se aprovisiona de bienes para cumplir con los procesos de venta a sus clientes.

Figura #11: Especificación de la categoría de patrones “Venta y Compra de Productos”.

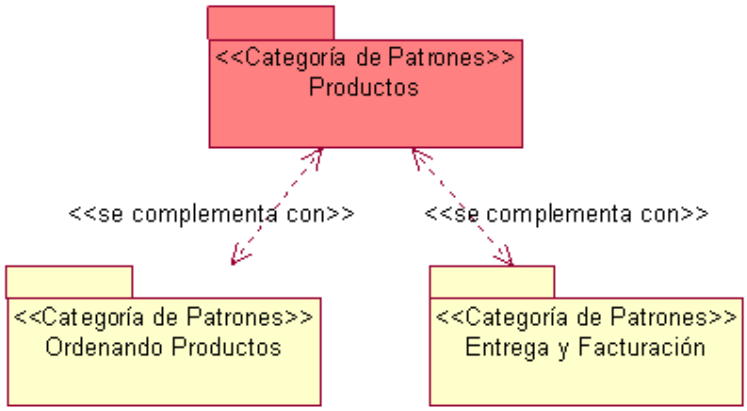

Figura #12: Subcategorías de la categoría central "Venta y Compra de Productos".

Intencionalidad: Los patrones de esta categoría generalizan artefactos del análisis que modelan requerimientos de información que tienen las organizaciones sobre sus productos, específicamente tipos y categorías de productos.

Los problemas del análisis de este dominio representados por esta categoría son, por ejemplo:

1. ¿Cómo organizar los artículos y servicios en tipos que representen sus atributos relevantes?
2. ¿Se requiere la definición de categorías de productos? ¿Qué tan estables son estas categorías?
3. ¿Se venden artículos? ¿Simples y compuestos?
4. ¿Se venden servicios?
5. ¿Se venden tanto artículos como servicios? En tal caso, ¿qué relaciones relevantes se presentan?

Terminología:
1. Producto: es un Producto Tangible o un Producto Intangible. Los productos intangibles se conocen típicamente como servicios.
2. Tipo de artículo: representa un conjunto de artículos que poseen características similares, en particular un mismo número de modelo, o marca o identificador generado por los proveedores de la empresa.
3. Tipo de artículo simple: es un tipo de artículo que, desde la perspectiva de los procesos del negocio, se maneja como una unidad indivisible.
4. Tipo de artículo compuesto: es un tipo de artículo que, desde la perspectiva de los procesos del negocio, se maneja como una composición de otros tipos de artículo que también aparecen dentro de los catálogos de la empresa o simplemente son relevantes para los procesos de producción.
5. Tipo de servicio: es un tipo de actividad que la empresa vende a sus clientes y por cuya realización obtiene algún tipo de beneficio que constituye una meta central en el negocio de la empresa.
6. Categoría de tipos de producto: es una categoría de tipos de artículos o de tipos de servicio que es relevante para los procesos del negocio y que puede tener una existencia temporal o indefinida. En el contexto de esta categoría se supone que un tipo de producto (artículo o servicio) solo puede pertenecer a una categoría, sin embargo en ciertos contextos organizacionales este supuesto no será válido. Es tarea del analista determinar la aplicabilidad de este supuesto fundamental.

Figura #13: Especificación de la categoría de patrones "Productos".

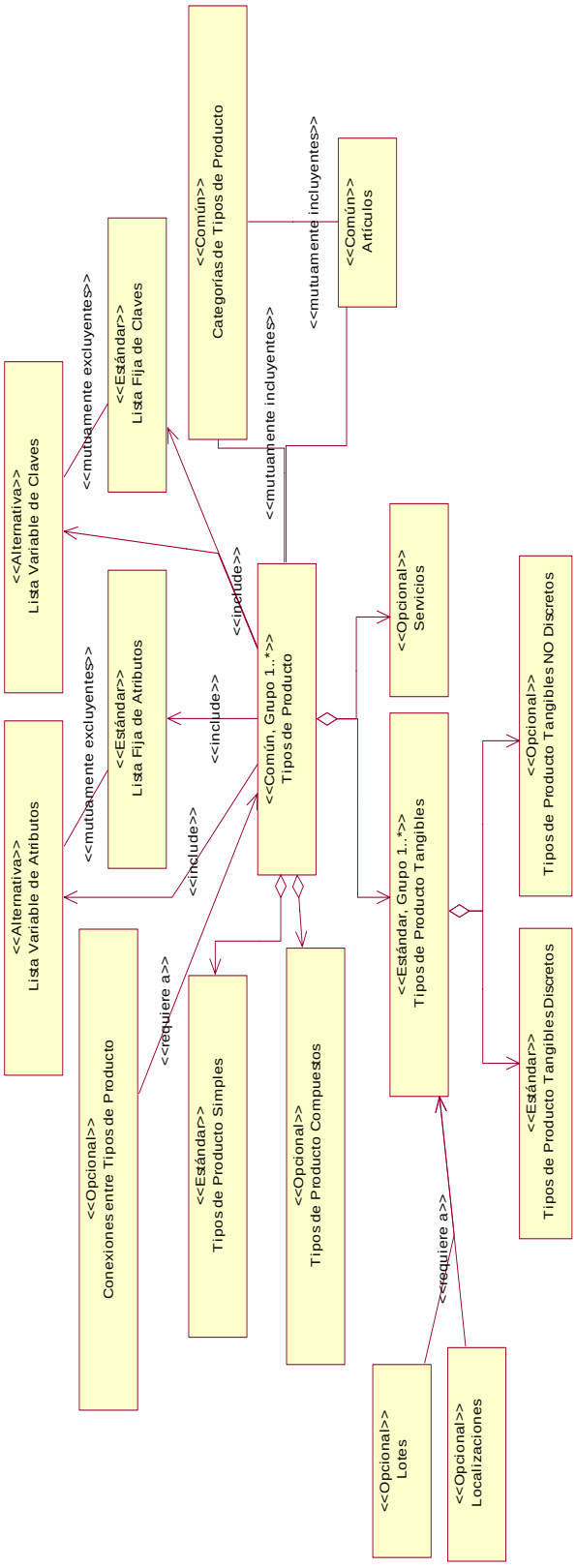

Figura #14: Características funcionales de la subcategoría central “Productos”.

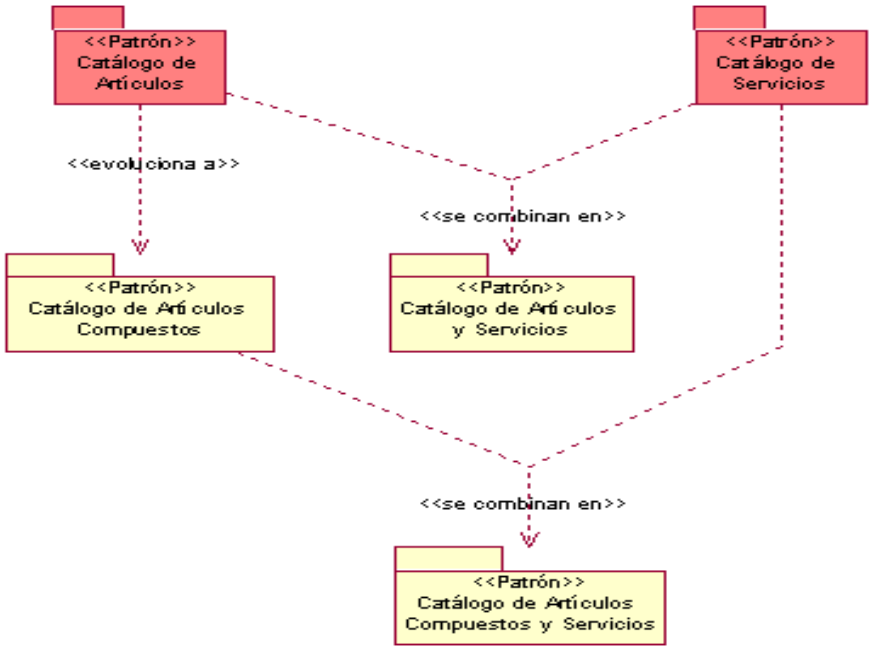

Figura #15: Patrones de la subcategoría central “Productos”.

Intencionalidad: Discernir claramente entre las entidades Artículo, Tipo de Artículo y Categoría (o Clase) de Tipos de Artículo visualizando los casos de uso, operaciones y atributos más comunes en este tipo de sistema.

Contexto: Pequeñas empresas comercializadoras y manufactureras cuyo negocio se centra en la venta de artículos completos y no partes. Ejemplos podrían ser pequeños supermercados y pequeñas empresas que producen alimentos básicos. En general no serán buenos ejemplos del contexto de aplicación de este patrón grandes supermercados, cadenas de supermercados que con frecuencia hacen ofertas especiales de paquetes de artículos, ni comercializadores de repuestos al por mayor o al detalle.

Aplicación al análisis:
1. El analista ajustará los modelos del patrón en colaboración con usuarios, clientes y expertos en el dominio centrándose en:
   1.1 Navegabilidad entre casos de uso.
   1.2 Atributos de los documentos y paneles de control.
   1.3 Operaciones de los documentos y paneles de control.
2. Para el análisis de variaciones más fuertes en el contexto de esta misma categoría de patrones, el analista considerará:
   2.1 La necesidad de un sistema de clasificación múltiple para los artículos. El patrón propone que cada tipo de artículo solo pertenece a una categoría, pero en algunos casos podría ser necesario que pertenezca a varias clases.
   2.2 La necesidad inmediata o tendencia evolutiva hacia tipos de artículos compuestos, para lo cual se aplicará el patrón "Catálogo de Artículos compuestos".
   2.3 La necesidad inmediata o tendencia evolutiva hacia incluir la venta de servicios en la empresa cliente, para lo cual se aplicará el patrón "Catálogo de Artículos y Servicios".
3. Para el análisis de variaciones más fuertes que desbordan el contexto de esta categoría de patrones, el analista considerará:
   3.1 La necesidad inmediata o tendencia evolutiva hacia incluir el control de proveedores de artículos, lo cual se trata en la categoría de patrones "Ordenando Productos".
   3.2 La necesidad inmediata o tendencia evolutiva hacia incluir el manejo financiero del inventario de artículos, lo cual se trata en la categoría de patrones "Contabilidad y Presupuesto".
   3.3 La necesidad inmediata o tendencia evolutiva hacia integrar el manejo del catálogo de productos con el control de los procesos de producción, lo cual se trata en la categoría de patrones "Procesos de Producción".

Figura #16: Especificación básica del patrón “Catálogo de Artículos”.

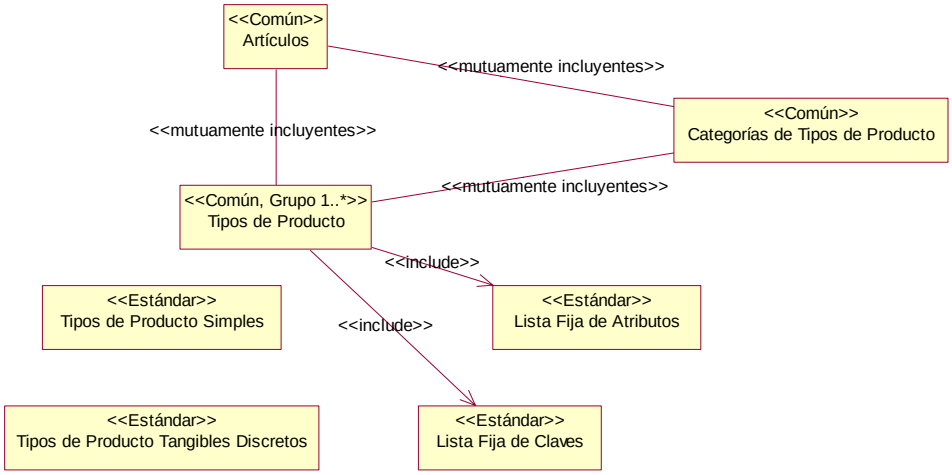

Figura #17: Combinación de características funcionales incluidas en el patrón “Catálogo de Artículos”.

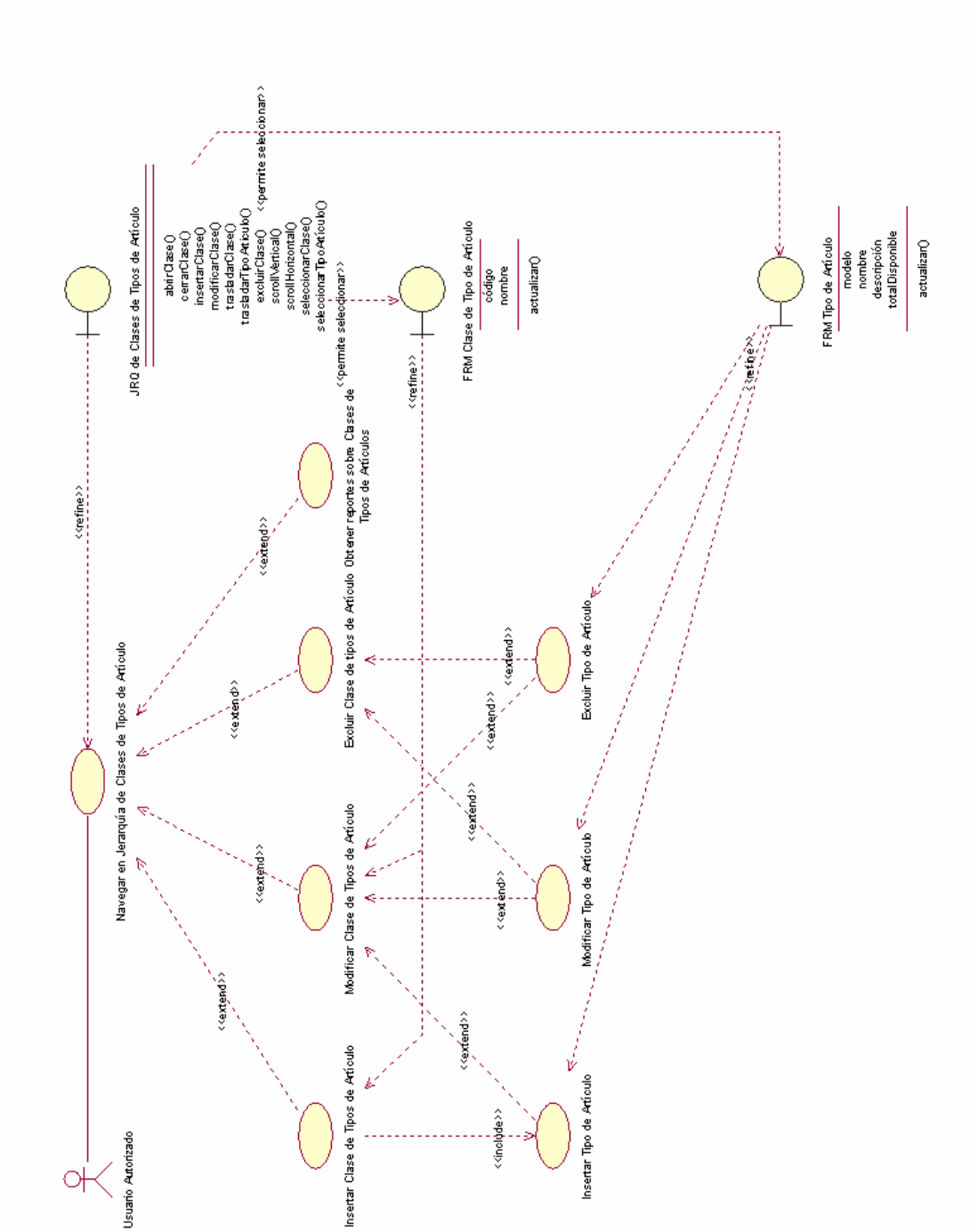

Figura #18: Casos de uso y objetos de frontera para la clasificación de Artículos del patrón "Catálogo de Artículos"[12].

[12] El prefijo "FRM" se usa para indicar que el objeto de frontera representa una ventana de formulario. El prefijo "JRQ" se usa para indicar que el objeto de frontera ofrece a los usuarios una organización jerárquica de los formularios que <<permite seleccionar>>.

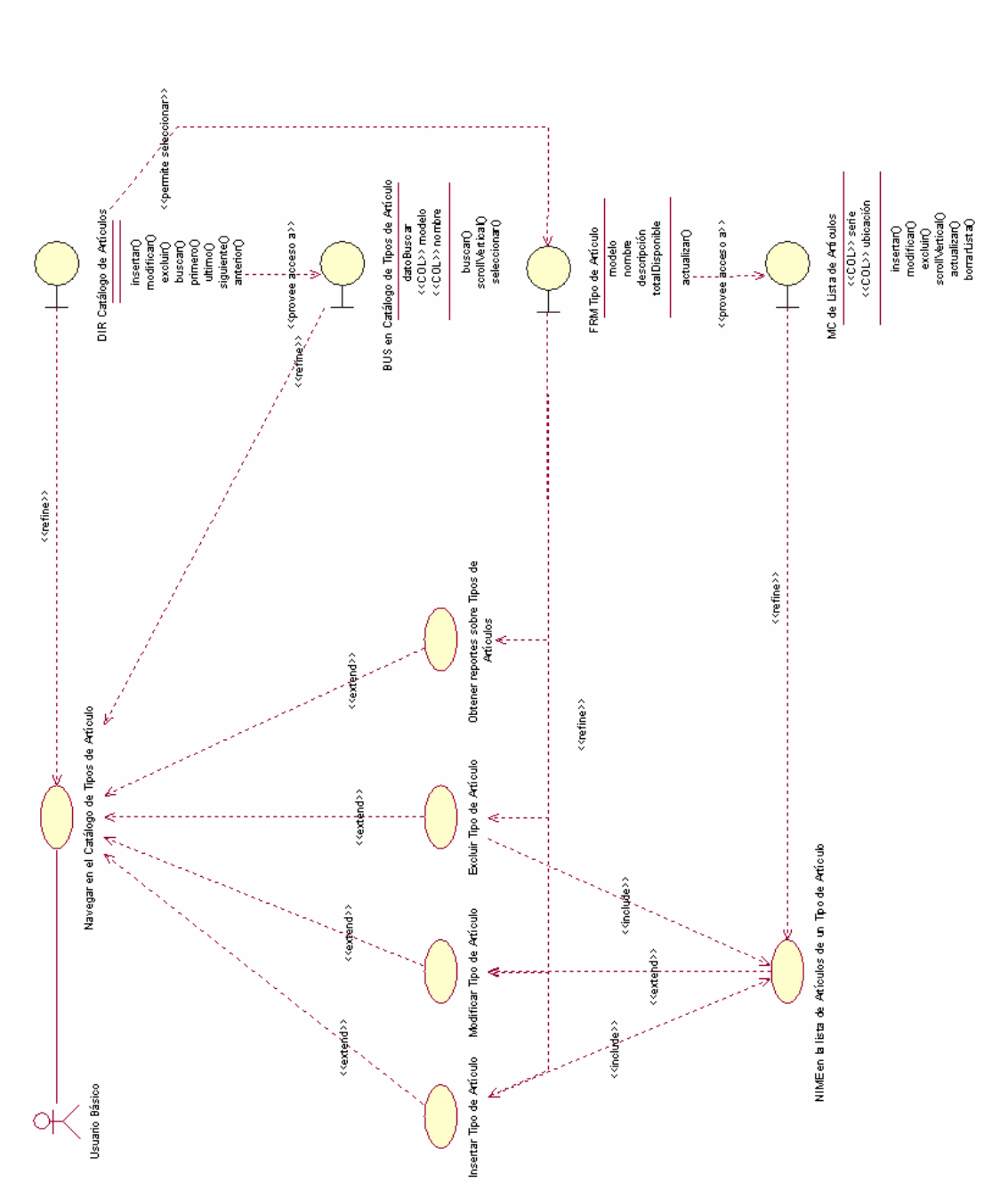

Figura #19: Casos de uso y objetos de frontera para tipos de Artículos del patrón "Catálogo de Artículos"[13].

[13] El prefijo "DIR" indica que el objeto de frontera ofrece a los usuarios una organización lineal de los formularios que <<permite seleccionar>>. El prefijo "BUS" representa un objeto de frontera que provee operaciones de búsqueda al usuario. El prefijo "MC" representa un objeto de frontera que ofrece a los usuarios una organización tabular de datos relevantes.

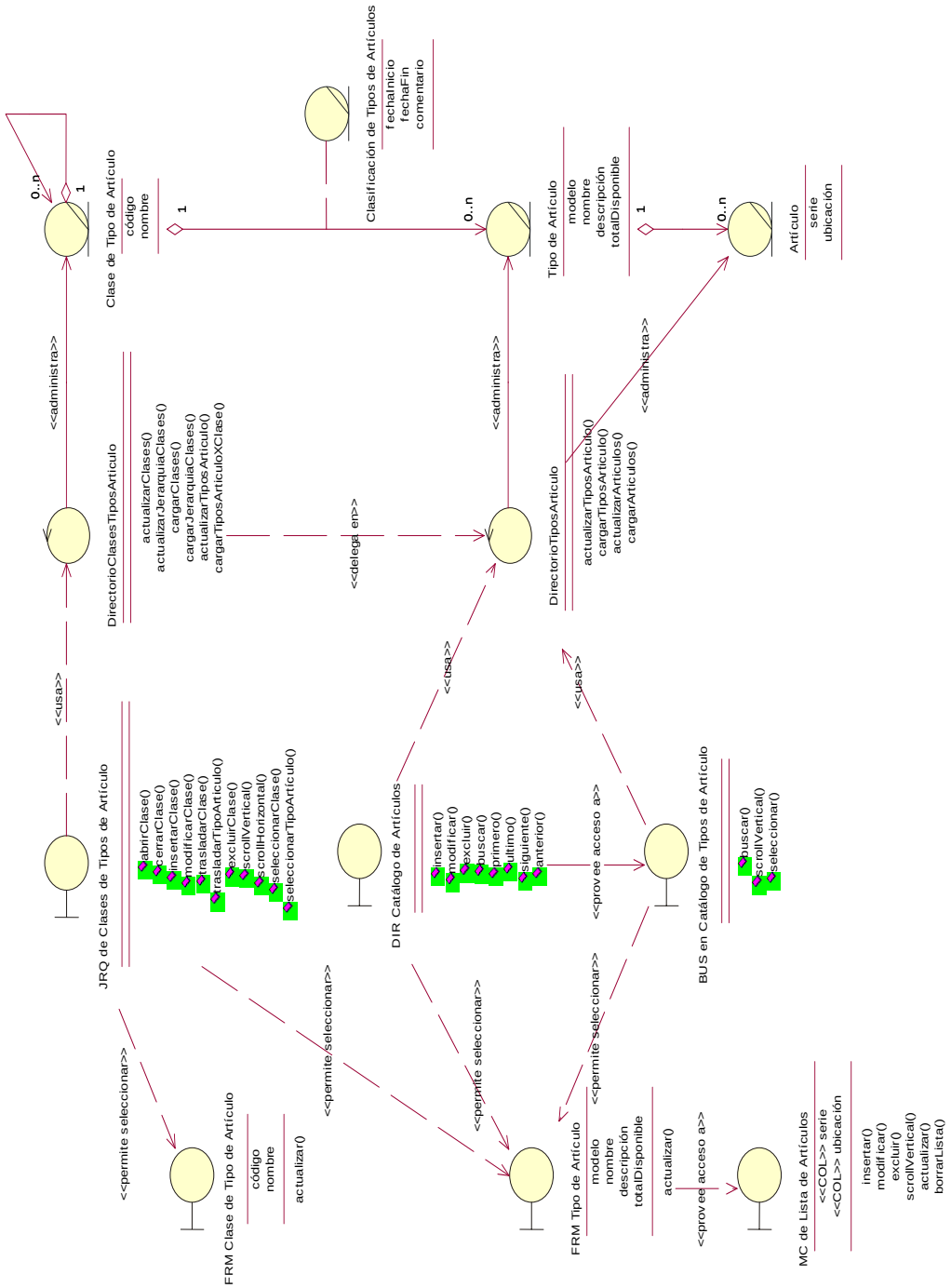

Figura #20: Modelo de Objetos de Análisis del patrón “Catálogo de Artículos”.

Intencionalidad: Discernir claramente entre las entidades Servicio, Tipo de Servicio y Categoría o Clase de Tipos de Servicio visualizando los casos de uso, operaciones y atributos más comunes en este tipo de sistema.

Contexto: En general empresas cuyo negocio es algún tipo de servicio, tales como servicios de comida, servicios de reparación o reconstrucción, servicios de salud o servicios de alquiler de artículos simples. En general no serán buenos ejemplos de contexto para la aplicación de este patrón las empresas que combinan su negocio entre la venta de artículos y la prestación de servicios, o empresas dedicadas al alquiler de artículos que, por sus características y las del servicio mismo, requieran conceptualizarse como artículos compuestos. Este último es el caso de empresas dedicadas al alquiler de computadores personales portátiles o sistemas de proyección para los cuales un control de partes puede ser relevante para controlar y mitigar el robo de partes, un riesgo natural para toda empresa dedicada al alquiler de artefactos.

Aplicación al análisis:
1. El analista ajustará los modelos del patrón en colaboración con usuarios, clientes y expertos en el dominio centrándose en:
1.1 Navegabilidad entre casos de uso.
1.2 Atributos de los formularios y paneles.
1.3 Operaciones de los formularios y paneles.
2. Para el análisis de variaciones más fuertes en el contexto de esta misma categoría de patrones, el analista considerará:
2.1 La necesidad de un sistema de clasificación múltiple para los servicios. El patrón propone que cada tipo de servicio solo pertenece a una clase, pero en algunos casos podría ser necesario que pertenezca a varias clases.
2.2 La necesidad inmediata o tendencia evolutiva hacia tipos de servicios compuestos.
2.3 La necesidad inmediata o tendencia evolutiva hacia incluir la venta de artículos en la empresa cliente, para lo cual se aplicará el patrón "Catálogo de Artículos y Servicios".
3. Para el análisis de variaciones más fuertes que desbordan el contexto de esta categoría de patrones, el analista considerará:
3.1 La necesidad inmediata o tendencia evolutiva hacia incluir el control de proveedores de insumos para los servicios, lo cual se trata en la categoría de patrones "Ordenando Productos".
3.2 La necesidad inmediata o tendencia evolutiva hacia integrar el manejo del catálogo de servicios con el control de los procesos de producción, lo cual se trata en la categoría de patrones "Procesos de Producción".

Figura #21: Especificación del patrón “Catálogo de Servicios”.

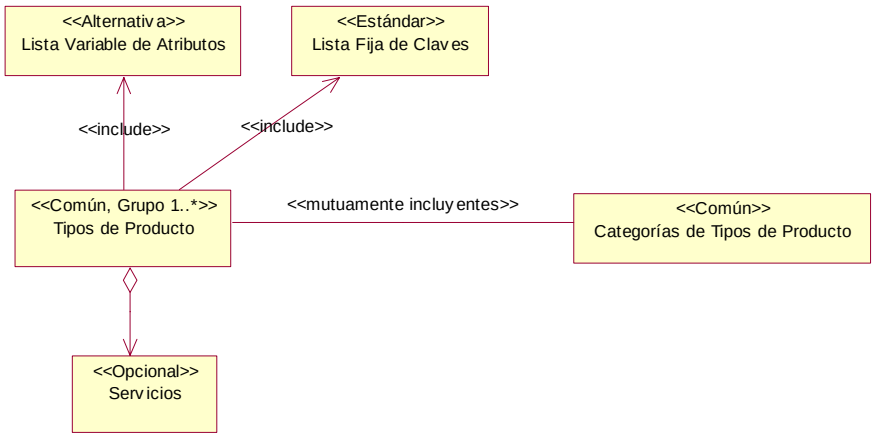

Figura #22: Combinación de características funcionales incluidas en el patrón "Catálogo de Servicios".

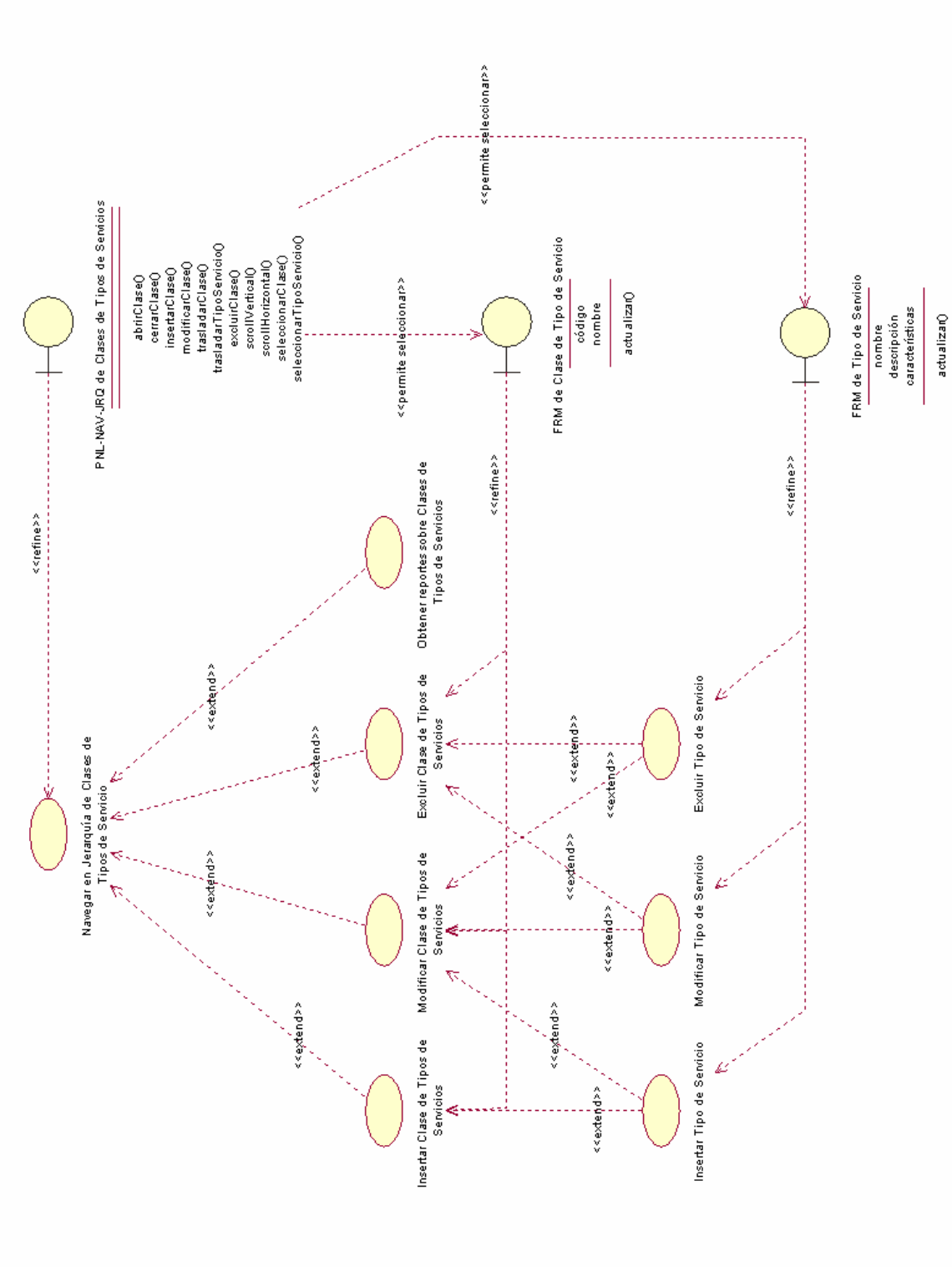

Figura #23: Casos de uso y objetos de frontera para la clasificación de Servicios del patrón "Catálogo de Servicios".

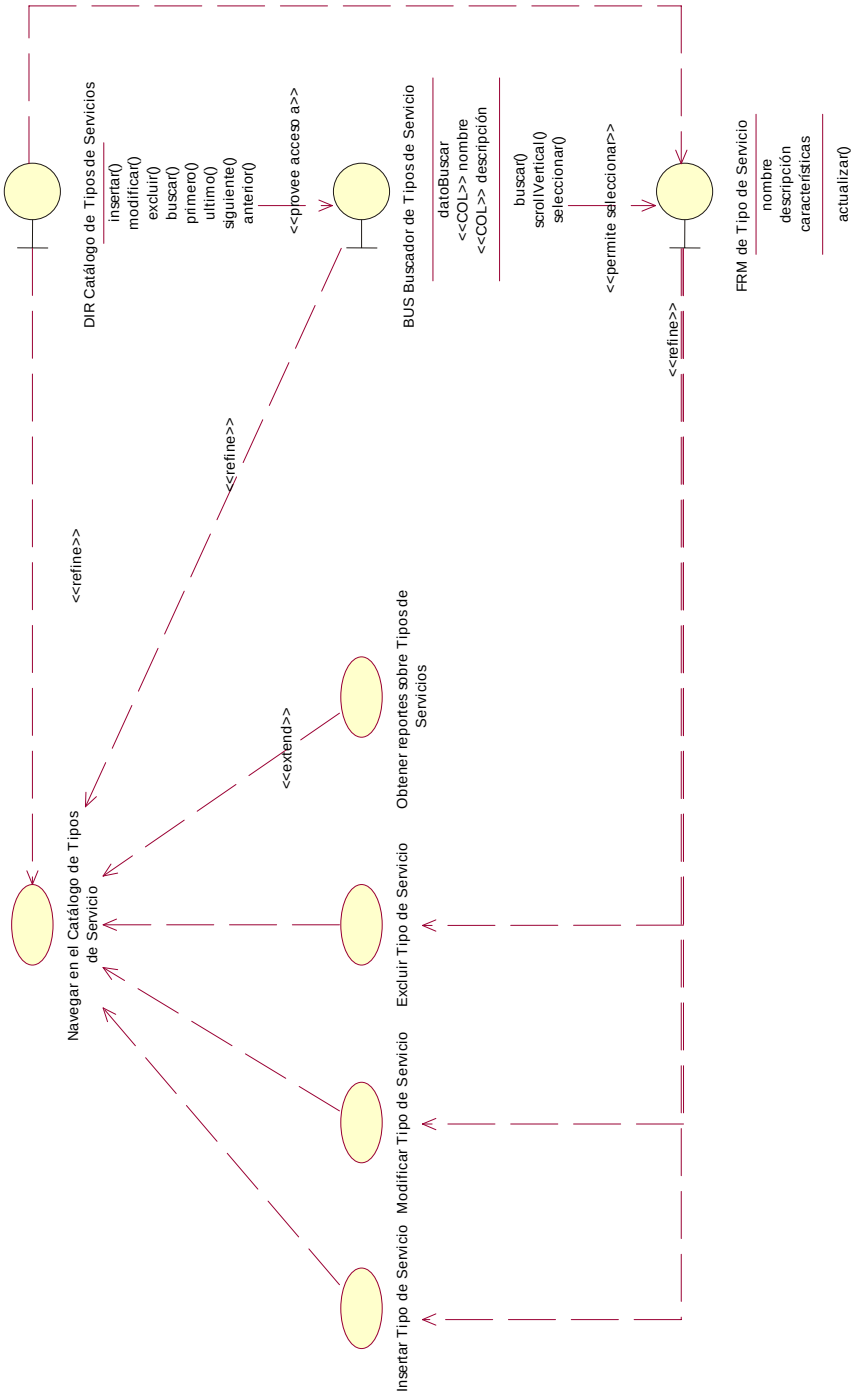

Figura #24: Casos de uso y objetos de frontera para tipos de Servicios del patrón “Catálogo de Servicios”.

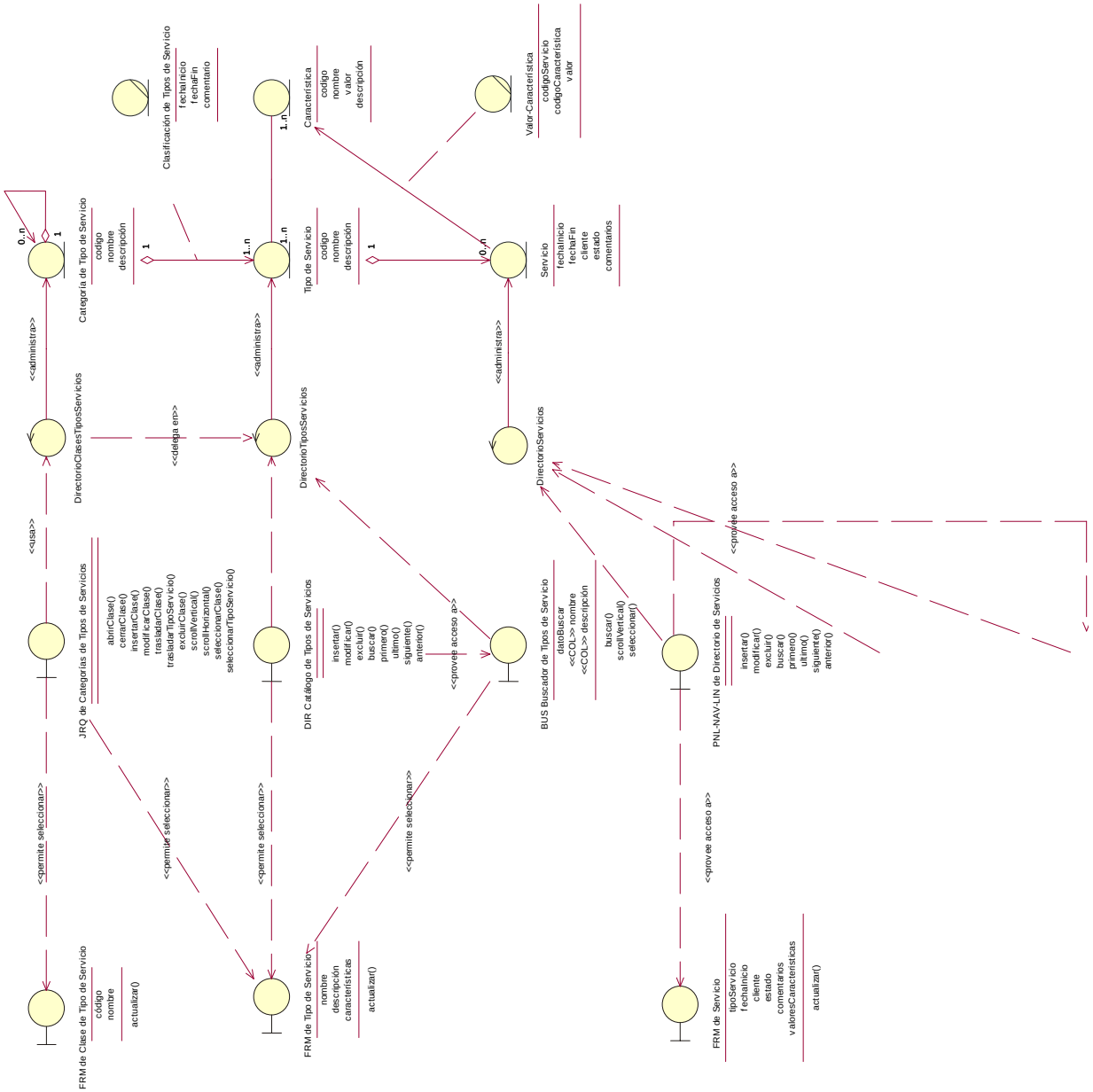

Figura #25: Vista parcial del Modelo de Objetos de Análisis del patrón “Catálogo de Servicios”.

5. Conclusiones y recomendaciones para la puesta en práctica

En resumen, se ha caracterizado el concepto de patrón citando a diversos autores de conocida trayectoria en el tema. Se han analizado los beneficios generales de usar patrones en el proceso de desarrollo de software, y más específicamente los beneficios de usar patrones de análisis, patrones arquitectónicos y patrones de diseño detallado de objetos. El Apéndice 1 describe con detalle la aplicación de ciertos patrones que se han descrito brevemente en el texto principal. Finalmente, el principal aporte que se hace en este informe es visualizar el uso de lenguajes de patrones para la organización del conocimiento en el contexto del desarrollo de software orientado a "software product lines" o "familias de productos de software" (FPS). Para esto se ha caracterizado el concepto de lenguaje de patrones citando el texto original en que se planteó por primera vez el tema [Alexander 1979]. Se ha reinterpretado el concepto de lenguaje de patrones en el contexto del desarrollo de software orientado a FPS caracterizando los *lenguajes de patrones de análisis de dominio*. Se ha discutido cómo se puede representar y organizar el conocimiento asociado a un lenguaje de patrones de análisis de dominio mediante un *lexicón de patrones* y finalmente se ha mostrado, mediante un ejemplo necesariamente esquemático, cómo se puede construir un lexicón de patrones para una familia de productos de software constituida por "Sistemas de información empresarial para procesos administrativos".

El uso de lenguajes de patrones de análisis de dominio en el desarrollo de software orientado a FPS merece unos comentarios adicionales dado que este tema no se ha tratado en la literatura especializada sobre patrones ni en la literatura especializada sobre FPS. En [Clements 2002] se usan patrones para representar y organizar conocimiento especializado en relación con la administración de FPS. En [Jacobson 1997], [Clements 2002] y [Gomaa 2004] se describe el uso de patrones de diseño y patrones arquitectónicos en el proceso de construcción de los artefactos reutilizables de la FPS y en la construcción de los productos de la FPS, no obstante en ninguno se enfoca el uso de patrones y lenguajes de patrones como una estrategia para representar, organizar y diseminar el conocimiento especializado que naturalmente se construye en una empresa desarrolladora de software al incursionar en un dominio de aplicación específico bajo el enfoque de FPS. Tanto en [Clements 2002] como en [Gomaa 2004] y en [Jacobson 1997] la administración de artefactos reutilizables enfatiza su valor práctico en el proceso de construcción de software, pero no profundiza en el conocimiento asociado a los artefactos, ni en la posibilidad o necesidad de administrar este conocimiento para facilitar el aprendizaje de la empresa desarrolladora de software como organización, ni en el valor práctico del conocimiento subyacente para la inducción de los desarrolladores novatos en una empresa desarrolladora de software que haya mantenido un enfoque de FPS durante varios años. Se ha mostrado que un lexicón de patrones de análisis de dominio podría servir para representar y organizar este conocimiento mediante una estructura que no solo facilita el aprendizaje a desarrolladores novatos sino también el aprendizaje de la organización como un todo, puesto que la propia estructura no obstaculiza la evolución del conocimiento representado. Por otro lado, el valor práctico de un lexicón de patrones para análisis de dominio en el proceso de construcción de software se puede lograr estableciendo conexiones entre los patrones, combinaciones de patrones y variaciones de patrones y los artefactos reutilizables de una FPS.

Aún cuando se ha argumentado y ejemplificado el uso de un lexicón de patrones para administrar el conocimiento naturalmente construido a lo largo de un proceso centrado en FPS, la demostración empírica de la utilidad de un lexicón de patrones en este contexto es un reto abierto que debería ser motivo de colaboración entre los centros de educación superior y la industria. En este sentido, se vislumbra una estrategia general que consiste en asignar a los centros de educación superior la responsabilidad de elaborar lexicones de patrones para diversos dominios de aplicación relevantes para la industria nacional de software; y a las empresas dejar la posibilidad de desarrollar los repositorios de artefactos reutilizables y particularmente los "frameworks" de objetos que les permitirían desarrollar productos de una FPS previamente acordada como relevante. De esta manera, los costos y los riesgos asociados con la sistematización del conocimiento del dominio (el análisis de dominio), que podría resultar difícil de afrontar para las empresas, sería absorbido por las instituciones de educación superior. Por la diversidad de su personal y su vocación para la investigación, las universidades están mejor preparadas para hacer frente al análisis de dominio. Por otro lado, el desarrollo de artefactos reutilizables y particularmente de "frameworks" de objetos requiere un conocimiento técnico y una perspectiva del mercado meta que concierne más a las empresas desarrolladoras de software. En una dinámica de colaboración de este tipo, la estructura de lexicón de patrones se podría afinar paulatinamente para responder efectiva y eficientemente a las necesidades de los desarrolladores de software bajo un contexto de FPS.

Finalmente, algunas recomendaciones para la adopción de patrones en los procesos de desarrollo de software, independientemente de si se hace bajo el enfoque de FPS o no:

1. Adquirir la bibliografía apropiada. La bibliografía usada en este informe es un subconjunto muy representativo de la bibliografía disponible.
2. Crear un directorio de sitios en Internet que pueden proveer información actualizada sobre el tema. El Apéndice 2 presenta una lista de algunos de los sitios más importantes que presentan información valiosa para muchas empresas.
3. Empezar estudiando los patrones de diseño detallado de objetos. El mejor punto de partida es [GoF]. Los patrones de diseño detallado de objetos han sido muy bien documentados y organizados en [GoF], [POSA1] y [POSA2], además de que son muy útiles en cualquier dominio de aplicación. En este sentido cabe enfatizar que en una empresa desarrolladora de software siempre se debería reconocer y, más aún, premiar el estudio y auto-aprendizaje de los desarrolladores de software (sin que por ello se deba dejar de lado la capacitación). Una hora diaria dedicada a la lectura y discusión de un patrón, en grupos de cuatro o cinco personas, puede ser una inversión con grandes beneficios en el corto plazo.
4. La inclusión de patrones en el proceso de desarrollo de software se puede planear en seis pasos:
   a. Estudio y discusión de patrones.
   b. Utilización de patrones en las actividades del diseño detallado y programación. Para ejemplos de aplicación de patrones de [GoF] al programar en JAVA™ y C#™ se puede consultar a [Shalloway 2004] y [Metsker 2004] respectivamente.
   c. Utilización de patrones en el diseño arquitectónico de sistemas de software (para esto se recomiendan [POSA1] y [POSA2]).
   d. Utilización de patrones en las actividades de verificación de los artefactos del diseño.
   e. Estudio de patrones de análisis para un dominio de aplicación específico. Esto implica de alguna forma que la estrategia de la empresa se incline a un enfoque de FPS.

   f. Elaboración de patrones de análisis para un dominio de aplicación específico. Para que esto pueda darse, se requiere la adopción explícita y el apoyo claro de los niveles gerenciales con respecto de un enfoque de desarrollo centrado en FPS.
5. Una vez que el estudio y la aplicación de patrones arquitectónicos y de diseño detallado han sido integrados en la cultura organizacional, se puede intentar incorporar patrones de interfaz humano-sistema o patrones de modelos de datos como una alternativa más simple a la de patrones de análisis para un dominio de aplicación específico.
6. Finalmente, dependiendo del grado de complejidad de la empresa desarrolladora de software, el estudio y la aplicación de patrones asociados con el proceso de aseguramiento de la calidad del software y el proceso administrativo de los proyectos de desarrollo de software, también podría ser muy beneficioso, pero probablemente no sea el mejor punto de partida para arrancar con la adopción de patrones en una empresa.

Bibliografía

[Alexander 1979] Alexander, Christopher. The Timeless Way of Building. Oxford University Press, Nueva York, 1979.

[Alur 2001] Alur, D., Crupi, J., Malks, D. Core J2EE Patterns (Best Practices and Design Strategies). Sun Microsystems Press, E.E.U.U., 2001.

[Booch 1996] Booch, Grady. Análisis y diseño orientado a objetos con aplicaciones. Addison-Wesley/Díaz de Santos, segunda edición, Delaware, 1996.

[Booch 1999] Booch, Grady; Rumbaugh, J.; Jacobson, Ian. El Lenguaje Unificado de Modelado. Addison-Wesley, primera edición en español, Madrid, 1999.

[Bruegge 2002] Bruegge, Bernd. Ingeniería de Software orientado a objetos. Prentice Hall, primera edición en español, México, 2002.

[Buschmann 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., Stal, M. Pattern-Oriented Software Architecture (A System of Patterns). John Wiley & Sons. West Sussex, Inglaterra, Octubre 1996. Conocido como **POSA 1**.

[Calderón 2003] Calderón Castro, Alan. Propuesta metodológica para la sistematización del conocimiento basado en patrones de modelos de requerimientos y en patrones de diseño de software, en grupos de ingenieros de software. Tesis aprobada por el Sistema de Estudios de Posgrado de la Universidad de Costa Rica, para optar por el grado de Magister Scientiae en Ciencias Cognoscitivas. San José, Costa Rica, 2003.

[Clements 2002] Clements, P. & Northrop, L. Software Product Lines: Practices and Patterns. Addison-Wesley, Boston E.E.U.U. 2002.

[Coad 1992] Coad, Peter. Object-Oriented Patterns. Communications of the ACM vol. 35(9), setiembre 1992.

[Coad 1997] Coad, Peter. Object Models (Strategies, Patterns and applications). Yourdon Press, New Jersey, 1997.

[Coplien 1992] Coplien, James. Advanced C++: Programming styles and idioms. Addison Wesley, 1992.

[Czichy 2001] Czichy, Thoralf. Pattern-based Software Development (An Empirical Study). Recibido directamente del autor cuyo correo electrónico es: thoralf@czichy.org.

[DeMarco 1978] DeMarco, Tom. Structured Analysis and System Specification. Prentice Hall Software Series, EEUU, 1978.

[Ericksson 2000] Eriksson, Hans-Erik & Penker, Magnus. Business Modeling with UML: Business Patterns at Work. John Wiley & Sons, 2000.

[Fernández 1999a] Fernández, Eduardo B. Building Systems Using Analysis Patterns. Proceedings PLoP 1999.

[Fernández 1999b] Fernández, Eduardo B. & Yuan, Xiaohong. An Analysis Pattern for Reservation and Use of Reusable Entities. Proceedings PLoP 1999.

[Fernández 2000] Fernández, Yuan & Brey. Analysis Patterns for the Order and Shipment of a Product. Proceedings PLoP 2000.

[Fowler 1997] Fowler, Martin. Analysis Patterns: Reusable Object Models. Addison-Wesley, E.E.U.U., 1997.

[Fowler 2002] Fowler, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley, E.E.U.U., 2004.

[Fowler] Martin Fowler. Sitio en Internet: http://martinfowler.com. Accedido por última vez el primero de febrero del 2006.

[Gamma 1995] Gamma, E., Helm R., Johnson, R. y Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company, 1995. Conocido como GoF.

[Gaertner 1999] Gaertner, Nathalie & Thirion, Bernard. Grafcet: an Analysis Pattern for Event Driven Real-time Systems. Proceedings PLoP 1999.

[GoF] ver [Gamma 1995].

[Gomaa 2004] Gomaa, H. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley, E.E.U.U. 2004.

[Grand 1999] Grand, Mark. Transaction Patterns: A Collection of Four Transaction Related Patterns. Proceedings PLoP 1999.

[Hamza 2002] Hamza, Haitham & Fayad, Mohamed E. A Pattern Language for Building Stable Analysis Patterns. Proceedings PLoP 2002.

[Hashler] Hashler, Michael. Analysis Patterns. Sitio en Internet: http://wwwai.wu-wien.ac.at/~hahsler/research/virlib_working2001/virlib/virlib.html. Accedido por última vez el primero de febrero del 2006.

[Hay 1996] Hay, David C. Data Model Patterns (Conventions of Thought). Dorset House Pub. New York, 1996.

[IDP] The Interaction Design Patterns Page. Sitio en Internet: http://www.visi.com/~snowfall/InteractionPatterns.html. Accedido por última vez el primero de febrero del 2006.

[Jacobson 1997] Jacobson, I., Griss, M. & Jonsson, P. Software Reuse: Architecture, Process and Organization for Business Success. Addison-Wesley, Massachussets, E.E.U.U. 1997.

[Jacobson 2000] Jacobson, I. Booch, G., Rumbaugh, J. El Proceso Unificado de Desarrollo de Software. PEARSON Educación, S.A. Madrid, 2000.

[Johnson 1992] Johnson, Ralph. Documenting frameworks using patterns. *Proceedings of OOPSLA*, 1992.

[Kuhn 1986] Kuhn, T.S. La Estructura de las Revoluciones Científicas. Editorial Fondo de Cultura Económica, México, 1986.

[Larman 1998] Larman, Craig. Applying UML and Patterns (An Introduction to Object-Oriented Analysis and Design). Prentice Hall, New Jersey, 1998.

[Luna 2000]

[Metsker 2004] Metsker, Steve. Design Patterns C#. Software Pattern Series. Addison-Wesley Professional, segunda edición, 2004.

[OMG] Object Managment Group. Sitio en Internet: http://www.omg.org. Accedido por última vez el 30 de enero del 2006.

[OMG-MDA] OMG MDA®: "Model-Driven Architecture" de OMG. Véase: http://www.omg.org/mda/executive_overview.htm. Accedido por última vez el 30 de enero del 2006.

[ORBs] Object Request Brokers. Sitio en Internet (actualizado en octubre del 2004): http://www.middleware.org/object/orb.html. Accedido por última vez el 30 de enero del 2006.

[PI] Patterns of Interaction: a Pattern Language for CSCW. Sitio en Internet: http://www.comp.lancs.ac.uk/computing/research/cseg/projects/pointer/pointer.html. Accedido por última vez el primero de febrero del 2006.

[PHP] Patterns Home Page. Sitio en Internet: http://hillside.net/patterns/Pattern. Accedido por última vez el primero de febrero del 2006.

[PLoP] Pattern Languages of Programming Conference. Sitio en Internet: http://st-www.cs.uiuc.edu/~plop/. Accedido por última vez el primero de febrero del 2006.

[POSA1] Véase [Buschmann 1996].

[POSA2] Véase [Schmidt 2000].

[Pree 1995] Pree, Wolfgang. Design Patterns for Object-Oriented Software Development. Addison-Wesley Pub., E.E.U.U., 1995.

[Schmidt 2000] Schmidt, D., Stal, M., Rohnert, H., Buschmann, F. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. John Wiley & Sons. West Sussex, Inglaterra, January, 2000. Conocido como **POSA 2**.

[Shalloway 2004] Shalloway, Alan & Trott, James R. Design Patterns Explained (A new perspective on Object Oriented Design). Software Pattern Series. Addison-Wesley Professional, segunda edición, 2004.

[Silverston 1996] Silverston, Len; Inmon, W.H.; Graziano, Kent. The Data Model Resource Book (A Library of Logical Data Models and Data Warehouse Desings). John Wiley & Sons, Inc. Nueva York, 1996.

[Sommerville 2002] Sommerville, Ian. Ingeniería de software. Addison-Wesley, sexta edición, México, 2002.

[Trejos 1999] Trejos, Ignacio; Luna, Antonio. El modelo de objetos: análisis y diseño. Club de Investigación Tecnológica, 1999.

[Vaccare 1999] Vaccare Braga1, Rosana T., Germano, Fernão S.R., Masiero, Paulo Cesar. A Pattern Language for Business Resource Management. Proceedings PLoP 1999.

[Yourdon 1978] Yourdon, Edward & Constantine, Larry. Structured Design (Fundamentals of a Discipline of Computer Program and Systems Design). YOURDON Press, Nueva York, 1978.

Apéndice 1: Directorio Abierto de Personas y Organizaciones

La intención de este apéndice es detallar el uso de algunos patrones en la construcción de un sistema hipotético. El DirAPO es un sistema basado en Internet para el manejo de clientes, proveedores y demás relaciones de una empresa y se usa en este informe para ilustrar el uso de algunos patrones de análisis, de diseño arquitectónico y de diseño detallado. Consecuentemente, el desarrollo del ejemplo abarcará la elaboración del modelo conceptual de objetos, del diseño arquitectónico y del modelo de clases de análisis.

1. El problema

Se trata de desarrollar el DirAPO no solo para satisfacer las necesidades de una empresa cliente en particular, sino que se plantea desde el comienzo una estrategia más amplia, según la cual se intenta obtener un producto que sea útil en varias empresas cliente (probablemente de distintos tipos) y que prevea o se adapte fácilmente a los cambiantes requerimientos de muchas empresas. Es en este contexto de variabilidad en términos de la evolución de los requerimientos y de la diversidad de contextos empresariales, donde el uso de patrones adquiere mayor relevancia. Esto es lo que se conoce como un proceso orientado a una familia de productos de software o "software product lines" en inglés (para una explicación más amplia de este enfoque se refiere al lector a [Jacobson 1997], [Clements 2002] y [Gomaa 2004]). Sin embargo, para mantener la exposición sencilla se omitirán en el análisis que sigue características propias de dicho enfoque.

En adelante se entiende por "empresa cliente" o simplemente "la empresa" como la organización donde el DirAPO será instalado en su fase de producción. Los requerimientos funcionales generales que se considerarán son:

1. Gestión del directorio de personas con que se relaciona la empresa.
   a. Las personas podrán ser físicas o jurídicas, clientes o proveedores.
   b. Si la empresa es una corporación, las personas podrán ser partes de la corporación o grupo empresarial, subsidiarias, divisiones, departamentos e inclusive grupos de trabajo.
   c. Las personas también podrán ser locales o extranjeras.
2. Gestión de contactos para cada persona.
3. Gestión de notificaciones para cada persona. Se entiende por notificación un texto relativamente corto que es relevante para la empresa en relación con alguna persona en el contexto de algún proceso de negocio.
4. Clasificación de las personas de acuerdo con las necesidades de la empresa.
5. Gestión de las relaciones relevantes entre las personas de interés para la empresa.
   a. Las relaciones podrán ser temporales o permanentes
   b. Las relaciones podrán ser de carácter jerárquico o no-jerárquico.
6. Si la empresa es una corporación, se requiere una administración integrada de los directorios de personas de todas las empresas de la corporación.
   a. Cada empresa de la corporación administrará su propio DirAPO.
   b. Un empleado autorizado de la corporación podrá tener acceso a los directorios de Personas de otras empresas de la corporación.

c. Usuarios no empleados de la corporación tendrán acceso limitado a los directorios de personas de todas las empresas de la corporación.

Los requerimientos no funcionales son:

1. Acceso eficiente al directorio corporativo o a los directorios locales través de Internet, tanto para usuarios internos de la empresa como para usuarios externos. La idea es que los representantes autorizados de clientes y proveedores de la empresa puedan acceder el directorio, consultar información pública y eventualmente actualizar su propia información.
2. Alta escalabilidad en cuanto a usuarios en acceso concurrente, debido a que es muy difícil prever cómo variará este factor de empresa a empresa.
3. Se deberá facilitar la incorporación de los sistemas de directorio de las nuevas empresas de una corporación. Los cambios que afecten la ubicación de los directorios locales serán transparentes para todos los usuarios.
4. Alta reutilizabilidad. Como parte del proceso se identificarán componentes que potencialmente serán reutilizables en otros proyectos orientados a procesos empresariales. Se pretende dar a estos a componentes un tratamiento especial, de tal manera que se logre reutilizabilidad máxima.
5. Alta flexibilidad. Dado que es una aplicación importante para los procesos de la empresa, es de esperar que nuevos requerimientos surjan una vez instalado, lo cual implica que el diseño deberá ser suficientemente flexible para incluir nuevos requerimientos con relativamente poco esfuerzo. Más aún, se espera que muchos cambios se den.
6. Alta adaptabilidad a cambios en la plataforma tecnológica. Esto significa que se espera que el diseño sea apto para cambios en el sistema de base de datos, en el mismo sistema operativo o en otros componentes de "middleware".

Los requerimientos usuales de control de acceso y en general de seguridad para una aplicación basada en Internet se omiten para simplificar la presentación. A continuación se construyen los modelos referidos utilizando algunos patrones relevantes.

2. Modelo conceptual de objetos

Para construir este modelo se han usado patrones de Fowler [Fowler 1997], Silverston [Silverston 1996] y parcialmente de Hay [Hay 1996]. La base ha sido el patrón "Accountability"[14] que aparece en [Fowler 1997].

"Accountability" se presenta en varias versiones que van de lo más concreto a lo más abstracto. Al final, el autor introduce variantes menos típicas pero que ha considerado relevantes. A lo largo de esta serie de versiones la intención del autor ha sido representar, en un modelo abstracto y flexible, tanto personas físicas como jurídicas que sostienen relaciones relevantes con la empresa. Estas relaciones pueden ser jerárquicas o no, están sujetas a reglas específicas y pueden cambiar con el tiempo. Es posible también representar las relaciones que dos personas tienen entre sí, si esto es relevante para la empresa. Las relaciones que Fowler ha considerado son binarias, por lo que en la figura A1-1 los "roles" o lugares que ocupan las organizaciones en una relación se han generalizado como "sujeto" y "objeto". Así por ejemplo, en la relación es "X es subsidiaria de Y", X ocupa el lugar del "sujeto" y Y del

[14] Se ha preferido mantener el nombre original en inglés a pesar de que se pierde la imagen inicial que invoca el nombre del patrón debido a la dificultad para encontrar una traducción apropiada.

"objeto". Se trata simplemente de marcar el lugar de la persona en la relación con base en la posición que ocupa en la estructura sintáctica de la frase que enuncia la relación. Se puede asignar una clase base a una persona, y a cada clase otra super-clase, por lo que las personas pueden pertenecer a un "linaje de clases". Una persona puede participar en muchas relaciones con otras personas. Las relaciones en sí pueden cambiar y pertenecen a una clase que establece las restricciones de participación para el "sujeto" y el "objeto". Uno de los tipos de restricciones que establece toda clase de relación consiste precisamente en cuáles clases de personas pueden participar como "sujeto" y cuáles como "objeto". Finalmente, una restricción general aplica a cada instancia de relación: el tipo de clase de persona de la persona que participa como "objeto" debe aparecer entre el conjunto de clases de persona que la clase de la relación admite como "objeto" y lo mismo sucede con el "sujeto" de la instancia de relación (esto es lo que aparece como comentario en el diagrama).

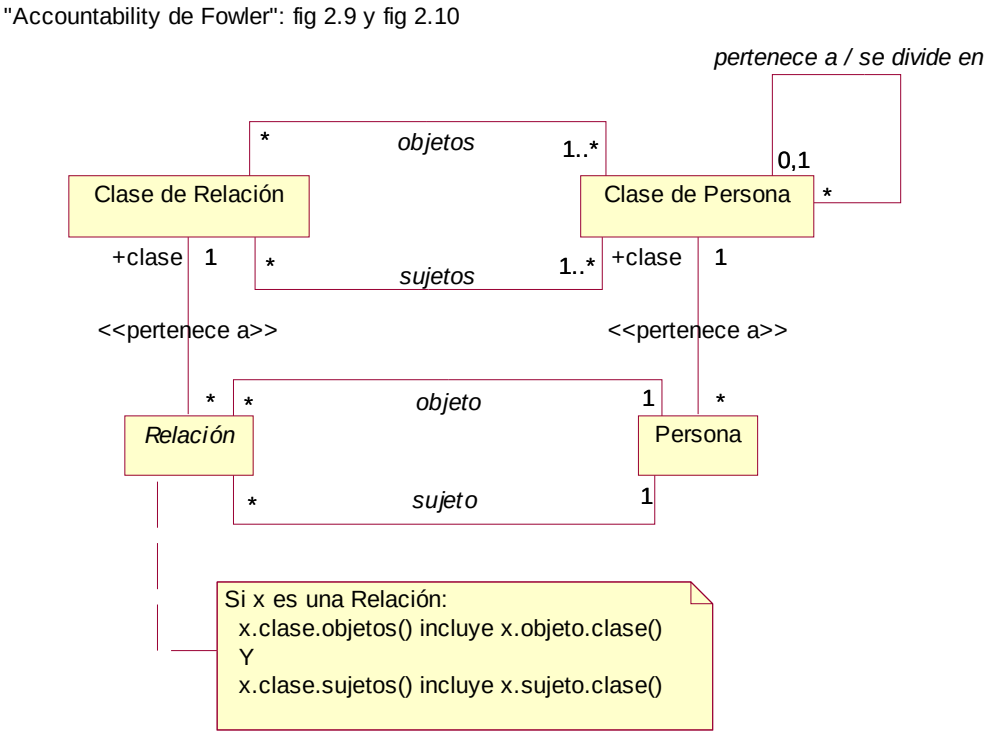

Figura #A1-1: "Accountability" de Fowler [Fowler 1997]

En síntesis, aunque Fowler no lo hace explícito, la intención de este patrón es proveer un modelo muy general para representar relaciones entre personas, considerando de manera especial que las relaciones entre personas físicas o jurídicas en sí pueden cambiar con el tiempo y a la vez suelen ser muy variadas. Esta intencionalidad es muy apropiada en el contexto del análisis del DirAPO. En la figura #AI-2 se muestra la contextualización de este patrón al análisis del DirAPO. Se han agregado características de "Relación de Persona" ("Party Relationship" en el original en inglés) de Silverston [Silverston 1996]. Particularmente la priorización y estado de una relación entre personas.

Para los requerimientos asociados a la administración de contactos y para el manejo de las direcciones de las personas se ha aplicado patrones de Silverston [Silverston 1996] y Hay

[Hay 1996] resultando el modelo de la figura #AI-3. El manejo apropiado de las direcciones debe considerar el hecho de que algunas personas pueden no residir en el mismo país que la empresa. Esto implica que deben considerarse diversos esquemas de dirección física. Hay[15] propone introducir el concepto de "Localizaciones geográficas" como una entidad compuesta, lo cual permitiría por ejemplo representar "País compuesto por Provincia compuesta por cantones compuesto por distritos" y también "País compuesto por Estado compuesto por Condado" (para mencionar esquemas de localización geográfica típicos de Costa Rica y E.E.U.U.). Tanto Hay como Silverston distinguen entre "Dirección" y "Localización geográfica" de manera semejante. Asimismo ambos autores introducen el concepto de "Condición de Dirección"; para este informe se ha adoptado la terminología del patrón "Definición de Dirección" de Silverston [Silverston 1996].

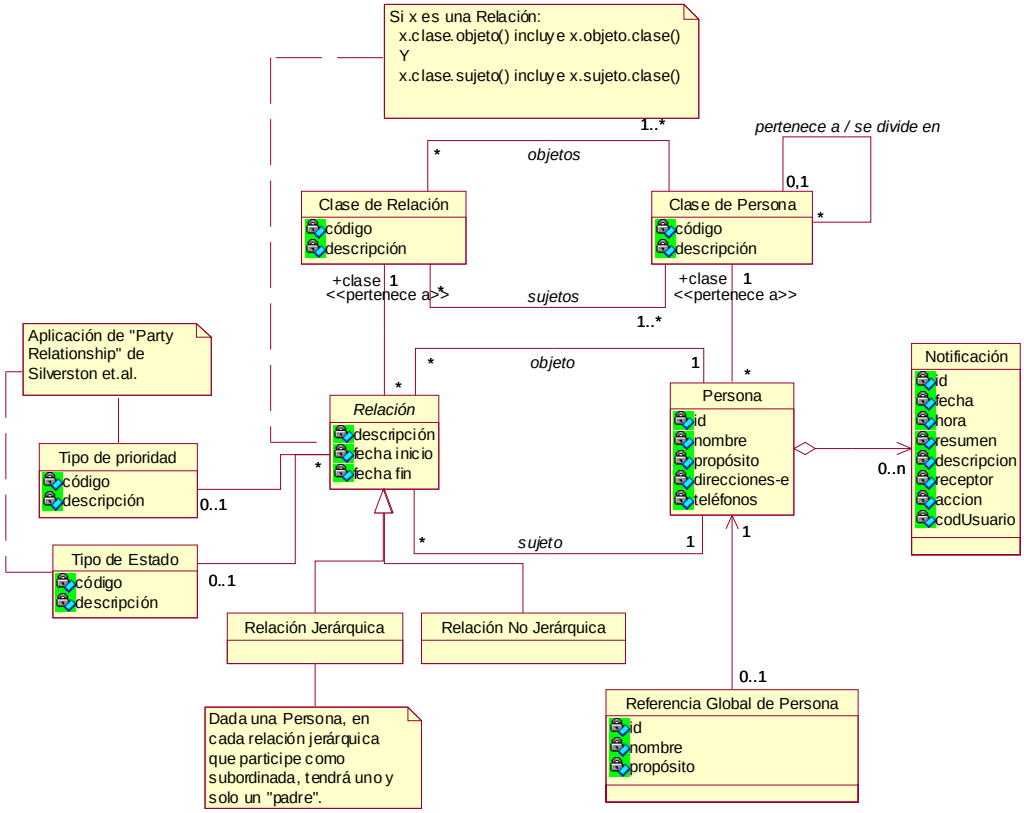

Figura #A1-2: Aplicación de "Accountability" de Fowler [Fowler 1997] y "Relación de Persona" de Silverston [Silverston 1996] al análisis de DirAPO.

Para introducir la administración de contactos para las personas se ha aplicado en la construcción del modelo el patrón "Definición de Mecanismo de Contacto" de Silverston [Silverston 1996].

[15] Silverston [Silverston 1996] también introduce este concepto, sin embargo plantea una estructura de solución más específica que la de Hay, por lo que se ha escogido la propuesta de este último autor.

Tanto las direcciones de una persona como los contactos pueden estar orientados a cierta función o propósito. Considérese por ejemplo la dirección de facturación y la dirección de envío en el caso de un cliente de la empresa. Análogamente, la persona o contacto encargada de tramitar la factura y el contacto en la empresa cliente que es el encargado de recibir la mercadería. Estos aspectos de la relación empresa-cliente cambian constantemente y por ello, en ambos casos, se adoptan datos de fecha. Esta es la justificación para los objetos "Condición de Contacto" y "Condición de Dirección".

En síntesis, la adopción de los patrones referidos en torno al manejo de las direcciones y al manejo de los contactos conlleva un modelo que se adapta fácilmente a los cambios en la relación de la empresa con clientes, proveedores u otro tipo de personas. Igualmente permite variar fácilmente los esquemas para registrar direcciones, lo cual constituye un aspecto básico para la internacionalización del sistema DirAPO.

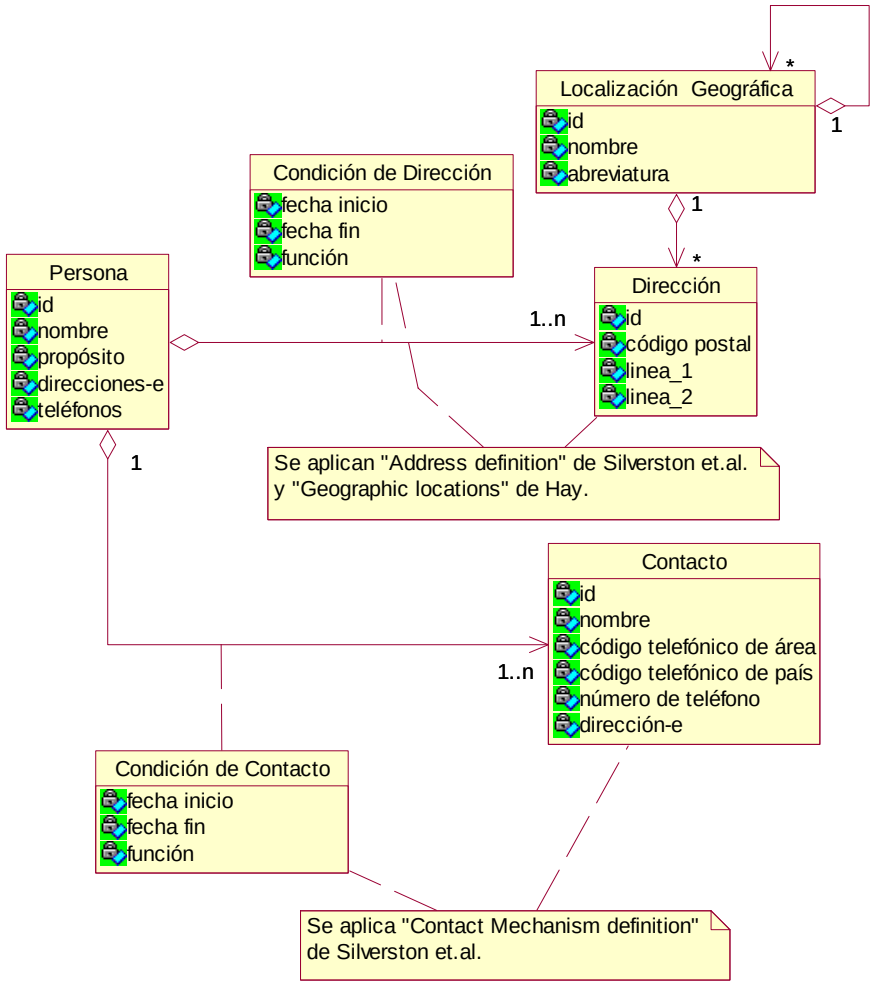

Figura #A1-3: Aplicación de patrones de Silverston [Silverston 1996] y Hay [Hay 1996] para el manejo de direcciones y contactos en el análisis de DirAPO.

La utilización de patrones de análisis de Fowler, Silverston y Hay ha permitido construir un modelo conceptual suficientemente detallado sin tener que hacer una indagación con clientes potenciales. Este modelo es una línea base para afinar el análisis en posteriores entrevistas con un cliente potencial. En un proceso de desarrollo orientado a una familia de productos de software permite considerar muchas variantes relevantes y negociar la confección de productos específicos con cada cliente en un contexto bien definido y estructurado. El modelo también puede ser útil para valorar un sistema legado que se pretenda incorporar al desarrollo de otro o simplemente mejorar.

3. Diseño arquitectónico

Bruegge señala que durante el diseño arquitectónico de software se debe poner especial atención en los requerimientos no funcionales (véase capítulo #6 de [Bruegge 2002]). Este autor también apunta que ciertos tipos de requerimientos no funcionales, por la propia forma en que se enuncian, ya sugieren la aplicación de algunos patrones. Bruegge alude aquí a patrones de [GoF] que por su intencionalidad coadyuvan en la construcción del diseño arquitectónico de un sistema. Ejemplo claro es "Fachada" que, basándose en el principio de "ocultamiento de información", facilita el desacoplamiento entre capas. La escogencia de "estilos arquitectónicos" es una tarea importante en la actividad de diseño arquitectónico y esto conlleva luego la aplicación subsecuente de otros patrones como "Fachada" para completar la tarea. Este tipo de conexión ("Patrón-X se puede implementar con Patrón-Y") entre patrones está bien documentada y aparece en muchos mapas de patrones entre los catálogos publicados.

En el caso del sistema DirAPO los requerimientos no funcionales #1, #2 y #6 indican la aplicación de "Multicapas" de [POSA1]. El requerimiento #3 indica el uso de "Broker" del cual se disponen versiones diferentes tanto en [POSA1] como en [Gomaa 04]. Para este informe se ha usado la versión de [Gomaa 04]. El requerimiento #5 sugiere la aplicación de "Reflexión" de [POSA1] que en realidad está implícito en "Accountability" de [Fowler 97] tal como se explica seguidamente. Del requerimiento #4 se deduce que se debe poner especial atención a la identificación de componentes reutilizables. El manejo de la clasificación de personas es susceptible de generalización por cuanto en los sistemas de información empresarial es muy común el manejo de jerarquías de clasificación. Piénsese por ejemplo en la clasificación de tipos de productos, en la clasificación de puestos y en la clasificación de cuentas contables. Se puede usar el patrón "Composición" para el diseño detallado de un componente genérico, altamente reutilizable, para el manejo de jerarquías de clasificación. La explicación se ha dejado para la sección "Modelo de clases de análisis" de este apéndice. A continuación se explican brevemente los patrones "Multicapas" y "Broker", así como su aplicación al caso de estudio.

Aplicación de "Multicapas"

Este es un patrón muy conocido y difundido que no requiere mayores explicaciones. Tan difundido está entre los ingenieros de software que trabajan en el dominio de los sistemas de información empresarial que se habla de "diseño multicapas" como una técnica básica. Por esta misma razón, el patrón ha sido presentado en infinidad de textos y publicaciones especializadas, así como sus variantes. Sin embargo, una especificación muy rigurosa aparece en [POSA1]. En dicho catálogo se recomienda el uso de "Multicapas" para casos en donde se requiere alta escalabilidad de los sistemas y adaptabilidad a cambios en la

plataforma de instalación. En cuanto al acceso eficiente de los usuarios (primer requirimiento), se puede lograr si al buen manejo de la escalabilidad se une el diseño de clientes livianos.

En la figura #A1-4 se muestra el diseño arquitectónico que incluye cuatro capas, a saber, los clientes, el "broker", la capa intermedia constituida por el componente principal del DirAPO y la capa de base de datos que abarca al adaptador (véase sección "Modelo de clases de análisis" de este apéndice) y al servidor de base de datos. En este caso se ha aplicado la variante "Multicapas suave" ("Relaxed Layers" en inglés [POSA1]), lo que significa que una capa puede "saltar" sobre su capa inferior inmediata, para acceder otra todavía más abajo (siguiendo la idea de la pila de capas). A continuación la explicación del 'broker'.

Aplicación de "Broker"

Sobre la intencionalidad de este patrón se ha señalado "El patrón arquitectónico *Broker* puede ser usado para estructurar sistemas distribuidos de software con componentes desacoplados que interactúan por medio de invocaciones de servicios remotos. Un componente de tipo 'broker' es responsable por coordinar la comunicación, como por ejemplo pasando solicitudes de servicios, así como también transmitiendo los resultados y excepciones generadas" (pág. 99 de [POSA1]). Por otro lado, Gomaa señala "Un 'broker' es un intermediario en las interacciones entre clientes y servidores. Los servidores registran ante el 'broker' los servicios que proveen. Los clientes pueden luego solicitar estos servicios vía el 'broker'. El 'broker' también proporciona **transparencia de localización**, lo cual significa que si el componente de servidor es trasladado a una localización diferente, los clientes no tienen que conocer del cambio y sólo el 'broker' debe reconocerlo" (pág. 260 de [Gomaa 2004]).

El requerimiento no funcional #3 del sistema DirAPO introduce la necesidad de un 'broker' para desacoplar los servidores locales de las empresas parte de una corporación entre sí y con los clientes. En este requerimiento se alude precisamente a la característica de "transparencia de localización" que explica Gomaa en la cita anterior. La idea es que los servidores del sistema DirAPO que requiera instalar cada una de las empresas de una corporación registran sus servicios ante el 'broker'. Luego, tanto los usuarios que son empleados de alguna empresa de la corporación como los que pertenecen a otras empresas (probablemente clientes o proveedores de alguna empresa de la corporación) tendrán acceso a los servicios a través del 'broker'. Si la localización del servidor de una de las empresas de la corporación cambia, ninguno de los clientes del 'broker' se verá afectado por el cambio, solo el 'broker' deberá ser notificado. Si eventualmente otra empresa se une a la corporación, inscribe su servidor ante el 'broker' y quedará accesible para todos los usuarios. Afortunadamente existen en la actualidad muchos componentes de "middleware" que implementan 'brokers' y que podrían ser utilizados en la implantación del DirAPO (véase por ejemplo [ORBs]). En [Gomaa 2004] se describen dos variantes de "Broker". "Broker Handle" está orientado a evitar que el 'broker' se convierta en un cuello de botella cuando la cantidad de solicitudes puede ser muy alta, y por ello sería la variante ideal para el caso del DirAPO.

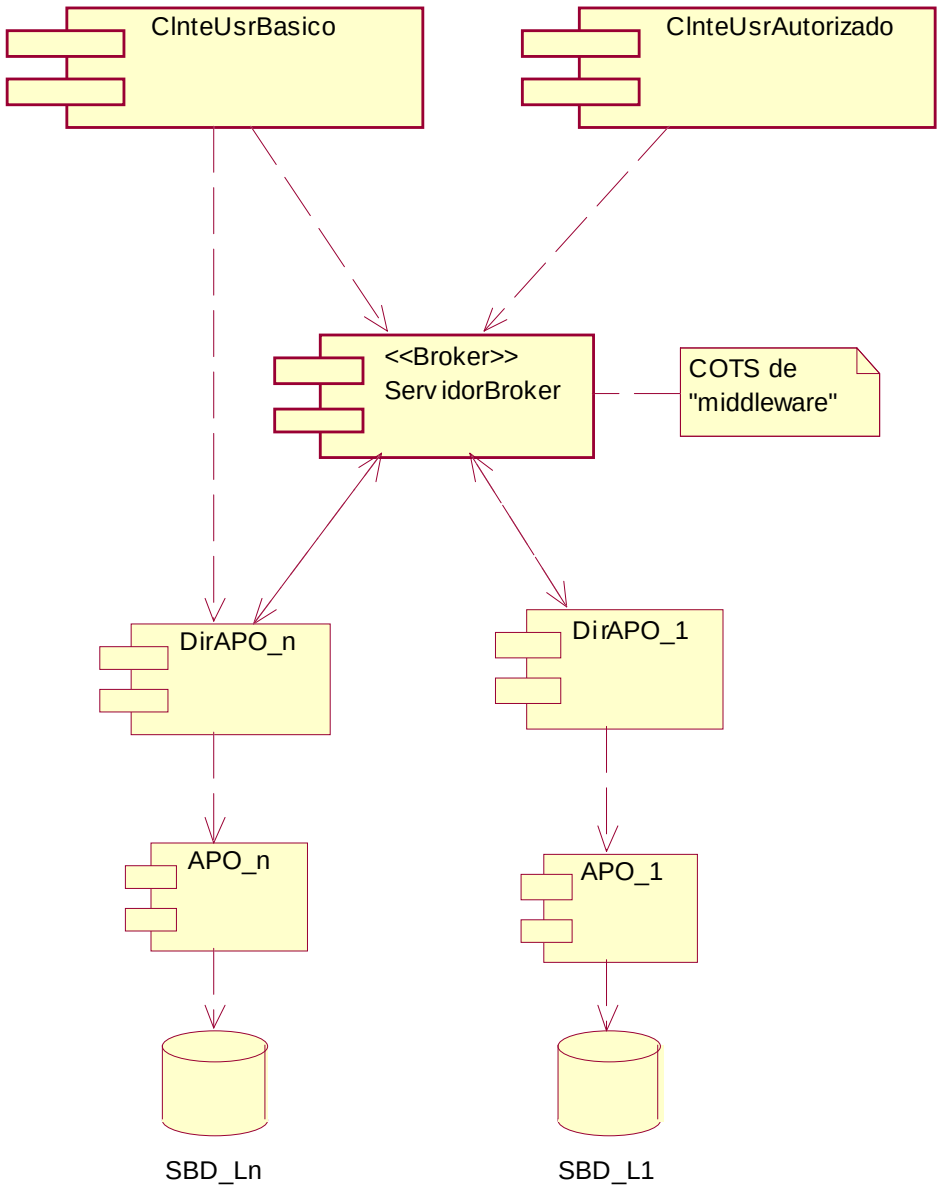

Figura #A1-4: Aplicación de patrones de “Multicapas” y “Broker” de [POSA1] y [Gomaa 2004] en el diseño arquitectónico de DirAPO.

Finalmente, en relación con la aplicación de estos patrones, cabe enfatizar que la de “Multicapas” introduce luego el uso de “Fachada” [Gof] para ocultar la complejidad de la capa intermedia al ‘broker’. Por otro lado “Proxy Remoto” hace todavía más invisible el ‘broker’ a sus clientes, tal como se sugiere en [POSA1]. Estos detalles sin embargo solo se notan cuando se desciende al nivel del diseño de objetos en la siguiente sección de este apéndice (allí se incluye también la explicación de “Reflexión” de [POSA1]).

En resumen, el diseño arquitectónico es considerado por muchos autores una actividad compleja y decisiva (véase por ejemplo capítulo #6 de [Bruegge 2002]). Hay coincidencia en que el diseño arquitectónico requiere de ingenieros experimentados. Aunque la enorme difusión de "Multicapas" nos permite verlo casi como obvio en la actualidad, sin embargo tomó muchos años de desarrollo tecnológico y metodológico concretar esta idea. Aún conociendo el patrón, la definición de la función principal de cada capa, dónde comienzan y terminan las responsabilidades de una capa, y cómo deben apoyarse en otras, no es un problema que se pueda dejar en manos de ingenieros novatos cuando se trata de sistemas complejos (véase pág. 50 de [POSA1]). Por tanto, el uso de patrones arquitectónicos puede mitigar riesgos importantes derivados de la falta de experiencia en el equipo de desarrolladores.

4. Modelo de clases de análisis

Para construir el modelo de clases de análisis se han usado "Composición", "Cadena de Responsabilidades", "Intérprete", una variante de "Fachada" y "Proxy" de [GoF], además de "Objeto de Valor" de Alur [Alur 2001]. Los patrones "Composición", "Cadena de Responsabilidades" e "Intérprete" han sido usados para la estructuración interna del componente de la capa intermedia de DirAPO, mientras que los demás patrones se han usado para afinar la utilización de "Multicapas" y "Broker" en el diseño arquitectónico.

"Composición" es un patrón muy apropiado para introducir el concepto de patrón de diseño tal como se plantea en [GoF]. Ya se ha dicho en la segunda sección de este informe cómo se plantea en [GoF] la noción de patrón, pero aquí cabe agregar que "Un patrón de diseño nombra, abstrae e identifica aspectos clave de una estructura de diseño frecuente y común, lo que lo hace útil para crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases participantes y las instancias, sus funciones y colaboraciones, así como la distribución de responsabilidades. Cada patrón de diseño enfoca un problema o tema de diseño orientado a objetos particular. Describe cuándo se puede aplicar, si puede aplicarse en el contexto de ciertas restricciones de diseño y las consecuencias y aspectos alternativos derivados de su uso." (pág. 4 de [GoF]). En cuanto al patrón "Composición", la intencionalidad según este catálogo es: "Componer objetos en estructuras de árbol para representar jerarquías totalidad-parte. 'Composición' permite a los clientes trabajar uniformemente con los objetos individuales y con las composiciones de objetos." (pág. 163 de [GoF]). La estructura general de la solución se muestra en la figura #A1-5.

"Composición" ha sido usado en dos ocasiones durante el diseño detallado de objetos de análisis del sistema DirAPO. Primero para lograr la mayor flexibilidad posible en el manejo de las localizaciones geográficas asociadas a las direcciones de las personas y luego para lograr la mayor flexibilidad posible en la representación de la jerarquía de clases de personas. La segunda aplicación es idónea si se decidiera construir un componente altamente reutilizable especializado en el manejo de jerarquías de clasificación.

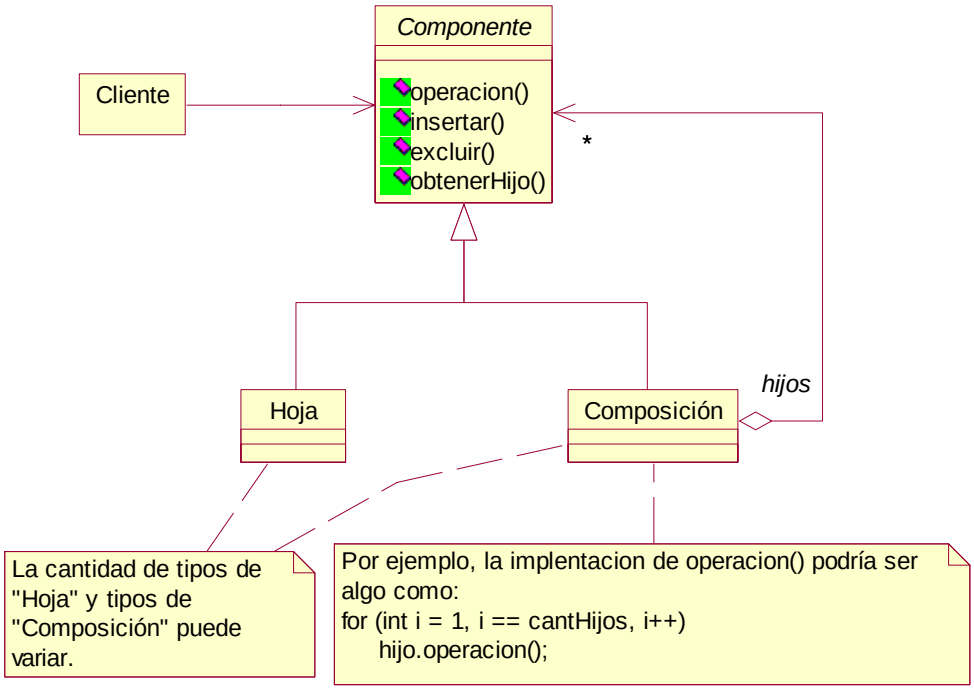

Figura #A1-5: Estructura de solución de “Composición” de [GoF].

En el primer caso, la representación de localizaciones geográficas, surge el problema de que, por ejemplo, el manejo apropiado de la composición jerárquica “país compuesto por provincia compuesto por cantón compuesto por distrito” requiere que no se permita agregar directamente un “distrito” a un país, pues el tipo de sus “hijos” en la estructura corresponde solamente a “provincia”. Esta situación es analizada en la sección de consecuencias, tal como es presentado “Composición” en [GoF]. Se sugiere allí que se deben introducir verificaciones, en tiempo de ejecución, para implantar tal tipo de restricciones. La solución en este caso ha sido introducir una función de verificación “vrfSubLoc()” en la clase “LocGeoCmp”. De esta manera cualquier otra jerarquía de composición de localizaciones geográficas, como por ejemplo, “país federado compuesto por estado compuesto por condado” (que sólo tiene tres niveles en comparación con la anterior que tiene cuatro) también puede representarse si se registran los tipos de localización geográfica correspondientes. En la figura #A1-6a aparece esta primera aplicación de “Composición”. Este patrón permite distintos tipos de variabilidad en el diseño. Por ejemplo, si en el futuro, cuando los sistemas de realidad virtual se hayan desarrollado lo suficiente, las empresas llegaran a tener una “dirección virtual” (distinta de la actual dirección electrónica) basada en áreas internacionales virtuales, se podría ampliar el diseño agregando clases como “LocVrtSmp” y “LocVrtCmp” derivadas de la actual “LocGeo”. El manejo de las “direcciones físicas” no cambiaría en nada.

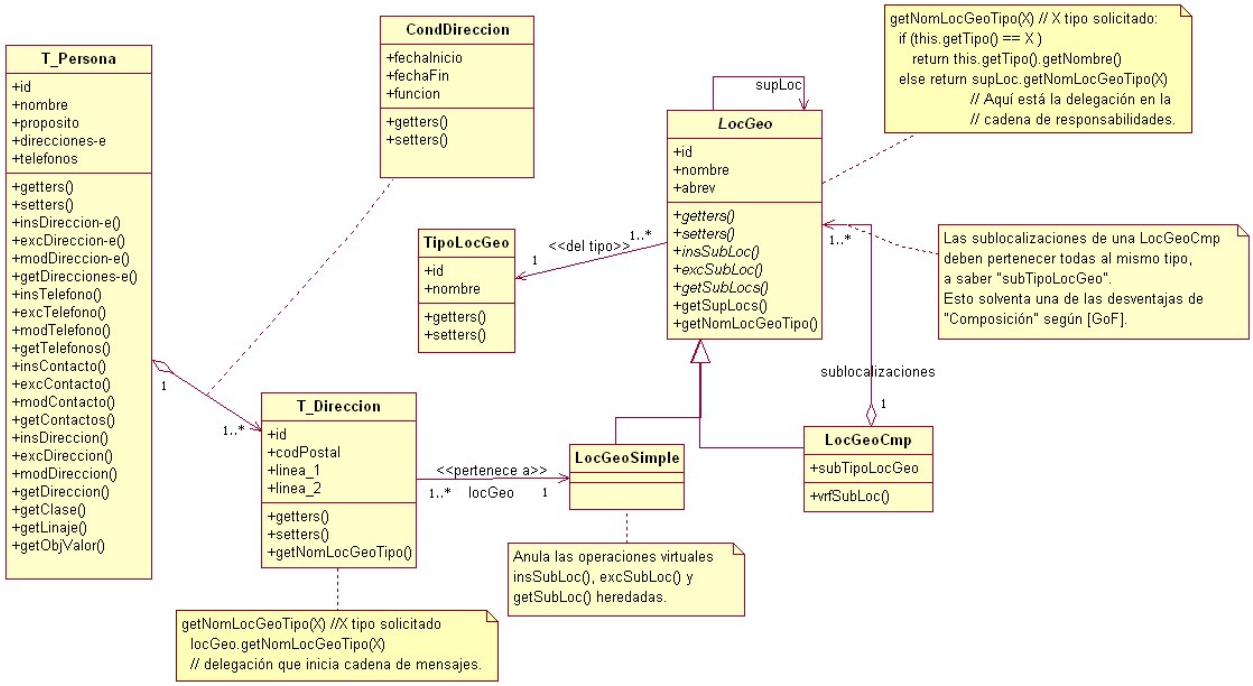

Figura #A1-6a: Aplicación de “Composición” y “Cadena de Responsabilidades” de [GoF] para representar direcciones basadas en jerarquías de composición de localizaciones geográficas.

La aplicación de “Cadena de Responsabilidades” la motiva el problema de que, si para una dirección de una persona, se quiere el nombre de alguna de las localizaciones geográficas a que pertenece, los datos están esparcidos en distintos objetos y no tiene sentido introducir en la clase “T_Persona” un método para cada tipo de localización geográfica, ya que el conjunto de tipos de localización geográfica varía con el país en que se ubica la persona en cuestión. Por ejemplo, en nuestro país se puede requerir consultar la provincia o el cantón asociado a la dirección de una persona X, pero en EEUU se puede requerir consultar el estado o la ciudad asociados a una dirección de una persona Y. Todos estos tipos se representan en el diagrama por medio de “TipoLocGeo”. Tenemos por tanto distintos tipos de localización geográfica dependiendo del país. Al requerirse el nombre de alguno de estos tipos de localización geográfica asociado a una dirección de una persona, el objeto que debe responder es la instancia de T_Persona, pero este objeto no posee la información requerida directamente. Por tanto debe delegar en la instancia de T_Dirección correspondiente. A su vez, este objeto tampoco “conoce” el dato requerido, por tanto debe delegar en la instancia de “LocGeoSimple” a que pertenece. Aquí inicia la cadena de mensajes típica de toda “cadena de responsabilidades”. La idea fundamental de este patrón es que el objeto que originalmente recibe la solicitud de información (en este caso la instancia de T_Persona) “no sabe” cuál es el objeto que puede responder y de hecho no necesita “saberlo” para poder responder.

El modelo de la figura A1-6a permite representar a dos personas con direcciones basadas en tipos de localización geográficas completamente diferentes. Este tipo de generalidad que es típica y fundamental para el desarrollo de familias de productos de software (ver la sección cuatro de este informe) conlleva problemas como el que se da al intentar recuperar la

dirección completa de una persona. Dado que la información está esparcida por varios objetos y que su estructura es diferente dependiendo del país en que se ubica la persona en cuestión, no funciona un método ingenuo que simplemente recorra los distintos objetos involucrados, puesto que ¿cuáles son y cuántos son?, además ¿en qué forma se debe estructurar la dirección requerida? (¿provincia, cantón, distrito y detalles o estado, ciudad, código ZIP y detalles?). Para resolver este problema, el patrón "Intérprete" puede ser muy útil. Desde la perspectiva de este patrón, cada dirección puede verse como el resultado de interpretar una "frase" de un lenguaje que permite representar distintos tipos de estructuras de direcciones siguiendo cierto conjunto de reglas bien definidas (la gramática). Las reglas en este caso son las reglas sintácticas generales de XML y las frases específicas son los esquemas de documento XML que corresponden a cada tipo o estructura de direcciones. Un método que "parsea" este tipo de esquema XML puede usar cada "nombre de elemento" (o "token" según "Intérprete") de un esquema XML para obtener el nombre del tipo de localización geográfica asociada a una dirección de una persona (mediante la operación "getNomLocGeoTipo()" de la clase T_Direccion). La estructura del esquema XML provee la estructura del tipo de dirección. La intencionalidad de este patrón es precisamente "Dado un lenguaje, definir una representación para su sintaxis junto con un intérprete que usa la representación para interpretar sentencias en el lenguaje". En la figura #A1-6b se muestran dos esquemas XML para la actual aplicación de "Intérprete".

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.org/amc"
      xmlns:tns="http://www.example.org/amc">
      <complexType name="tDireccionCR">
            <sequence>
                  <element name="provincia" type="string"></element>
                  <element name="canton" type="string"></element>
                  <element name="distrito" type="string"></element>
                  <element name="lineaDetalle" type="string" minOccurs="1" maxOccurs="unbounded"></element>
            </sequence>
      </complexType>
</schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.org/amc"
      xmlns:tns="http://www.example.org/amc">
      <complexType name="tDireccionEEUU">
            <sequence>
                  <element name="estado" type="string"></element>
                  <element name="ciudad" type="string"></element>
                  <element name="zip" type="string"></element>
                  <element name="lineaDetalle" type="string" minOccurs="1" maxOccurs="unbounded"></element>
            </sequence>
      </complexType>
</schema>
```

Figura #A1-6b: Esquemas XML para la aplicación de "Intérprete".

El segundo caso de aplicación de "Composición" es más directo y se representa en la figura #A1-7. En este caso se muestra un diseño específico y empotrado en el de DirAPO, pero si se quisiera independizar el manejo de jerarquías como un componente separado para fines de reutilización, el diseño de dicho componente sería el que aparece en la figura #A1-8. La idea es que cada vez que un subsistema requiera los servicios del componente "Clasificador

Genérico", éste se instanciaría con los datos de las clases específicas que el subsistema cliente requiere representar. El componente clasificador tendría que obtener de la base de datos los datos de las clases de la jerarquía que se requiere manejar, por lo que tendría su propio objeto para conectarse a la base de datos. Debido a que este componente podría recibir una solicitud de un subsistema remoto, sería conveniente incluir objetos de valor (ver adelante) para administrar eficientemente el traslado de datos entre el clasificador y su cliente.

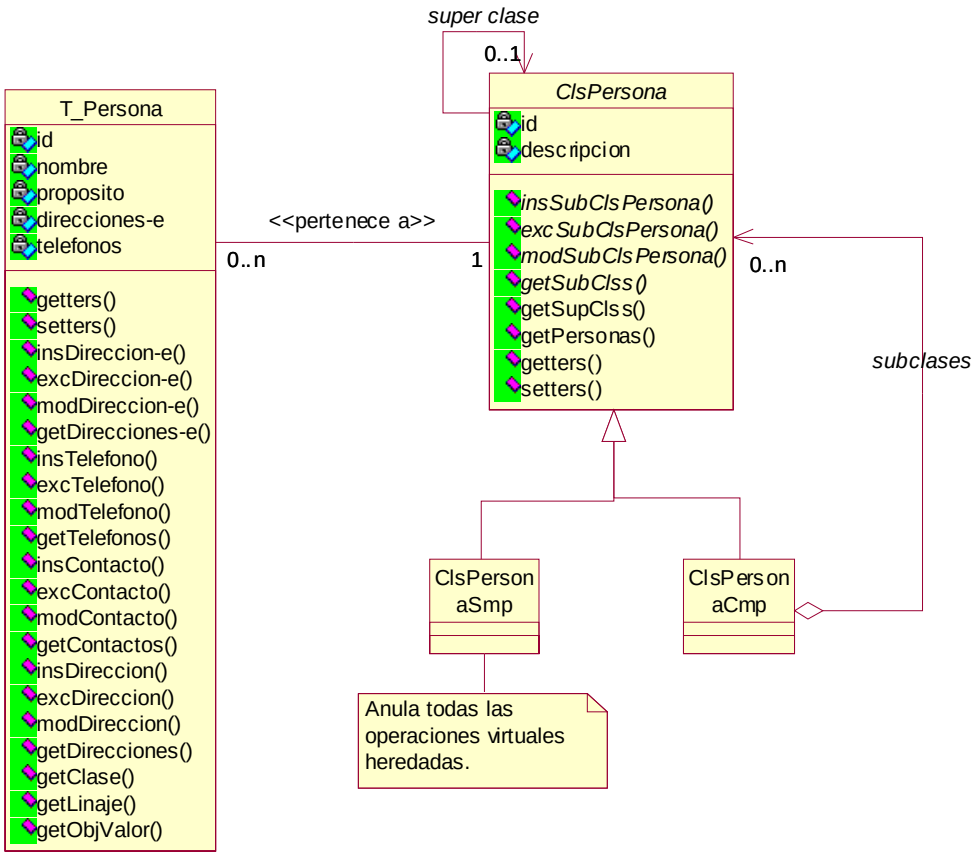

Figura #A1-7: Aplicación de "Composición" de [GoF] para representar jerarquías de clasificación de personas.

En las figuras #A1-8 y #A1-9 se muestra el uso de dos patrones que, como se ha dicho, permiten afinar el diseño arquitectónico basado en "Multicapas". Se trata de "Objeto de Valor" y "Fachada".

"Objeto de Valor" lo explica Alur [Alur 2001] como una buena práctica al usar J2EE (Java 2 Enterprise Edition), la plataforma especificada para el desarrollo de aplicaciones empresariales distribuidas de Sun Microsystems. Sin embargo este patrón es fácilmente generalizable (y probablemente ya está documentada una versión general). La idea es minimizar la cantidad de invocaciones remotas en los sistemas distribuidos destinadas a pasar datos, sustituyéndolas por unas pocas invocaciones que pasan grupos de datos relacionados.

Típicamente se pasan todos los datos de una entidad compleja o, aún más, compuesta (véase "Objeto de Valor Compuesto" en [Alur 2001]). Además, se busca un mecanismo de transmisión eficiente, por lo que en la actualidad se asocia la aplicación de este patrón con el uso de XML para codificar el estado de un objeto y pasarlo "a través de la red". Por tanto, este patrón permite afinar un diseño basado en "Multicapas" al hacer más eficiente el mecanismo de trasiego de datos entre las capas, que bien podrían residir en estaciones remotas, como es el caso del sistema DirAPO.

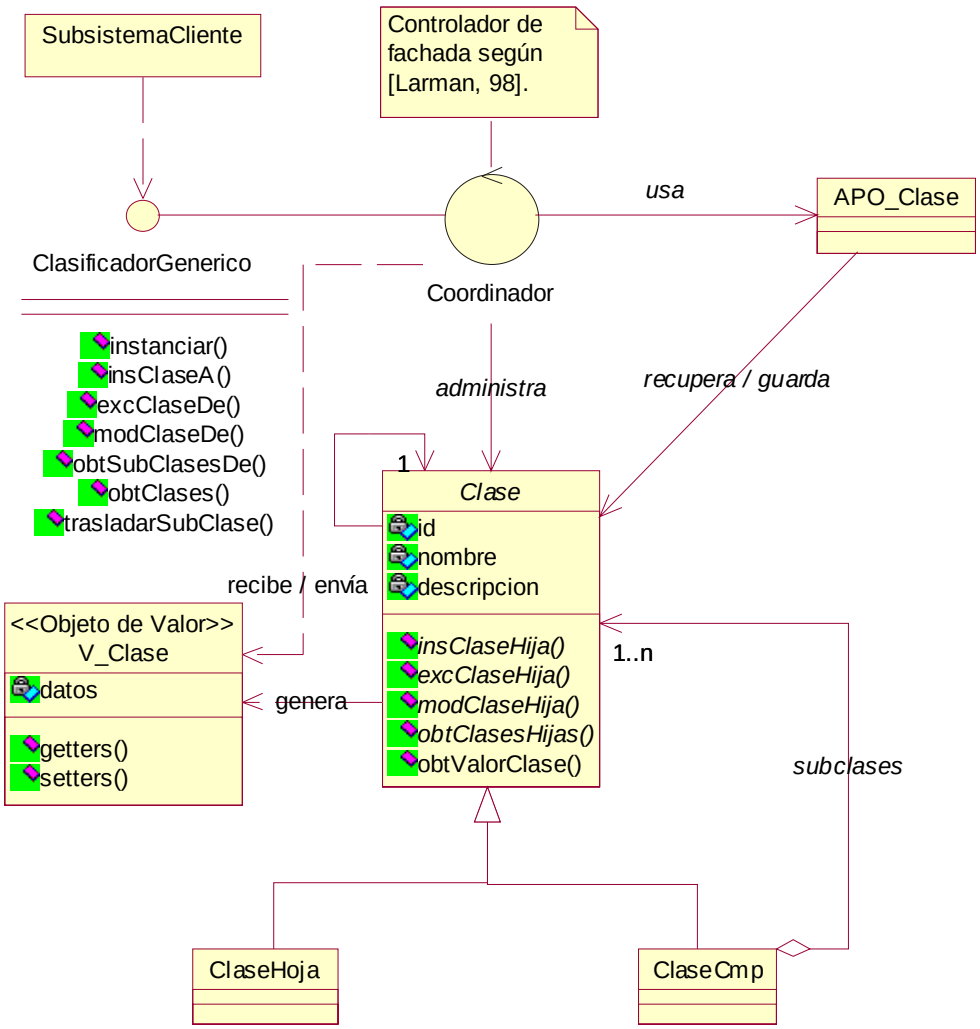

Figura #A1-8: Aplicación de "Composición" de [GoF] para representar jerarquías de clasificación en general, encapsuladas en un componente especializado.

"Fachada" es un patrón que aparece explicado ampliamente en [GoF]. Este es un patrón muy utilizado para desacoplar subsistemas, capas o componentes. "Proveer una interfaz unificada para un conjunto de interfaces en un subsistema. Fachada define una interfaz de más alto

nivel que facilita el uso del subsistema" (pág. 185 de [GoF]) es la intencionalidad que se adscribe a este patrón. La variante "Controller" descrita en [Larman 1998] (o "Controlador de fachada" en alusión clara a su origen) indica "Asígnese la responsabilidad de responder a mensajes de eventos de sistema a una clase según alguna de las siguientes posibilidades:

1. La clase representa a todo el sistema (controlador de fachada).
2. La clase representa al negocio u organización (controlador de fachada).
3. La clase representa algo en el 'mundo-real' que es activo (por ejemplo representa el cargo de una persona) y que puede estar involucrado en la tarea aludida (controlador de "rol").
4. La clase representa un administrador artificial de todos los eventos de sistema de un caso de uso (controlador de caso de uso)." (pág. 206 de [Larman 1998]).

Figura #A1-9: Aplicación de "Objeto de Valor" de [Alur 2001] y "Controlador de Fachada" de [Larman 1998].

En nuestro caso, los "coordinadores" de las figuras #A1-8 y #A1-9 pertenecen al primera opción prescrita por Larman. Los objetos "APO_Clase" y "APO_Persona" que aparecen respectivamente en las figuras referidas se introducen para, por un lado ocultar a los objetos de la capa intermedia los detalles del acceso a la base de datos, pero también para manejar "cachés" de los objetos respectivos, así como la transformación de objetos en tuplas y viceversa. Esta es una recomendación de Larman (págs. 455 a 465 de [Larman 1998]) que se basa en un patrón atribuido a otro autor. Estos objetos corresponden con los componentes "APO_1" a "APO_n" de la figura #A1-4 que presenta el diseño arquitectónico del sistema DirAPO[16].

En cuanto al uso de "Proxy" de [GoF] se especifica en este catálogo que "Provee un substituto de un objeto para controlar el acceso a él mismo" (pág. 207 de [GoF]). Luego se agrega "Un Proxy Remoto provee un representante o substituto local de un objeto en un espacio de direccionamiento diferente". En [POSA1] se recomienda que la comunicación mutua entre clientes y un 'broker' se dé a través de substitutos locales de cada lado para "...proveer transparencia, en el tanto en que un objeto remoto aparece ante el cliente como local. Más en detalle, los 'Proxies' permiten el ocultamiento de aspectos de implantación a los clientes tales como:

- Los mecanismos de comunicación entre-procesos que se usan para la transferencia de mensajes entre clientes y 'brokers'.
- La creación y destrucción de bloques de memoria.
- El 'marshalling' (conversión de tipos de datos) de parámetros y resultados." (pág. 104 y 105 de [POSA1]).

En los sistemas de software para 'brokers', los substitutos típicamente ya están implantados en los componentes que se instalan a ambos lados (cliente y 'broker'). En la figura #A1-10 se muestra en términos generales este uso de "Proxy" para afinar la aplicación del patrón "Broker" en el diseño arquitectónico de un sistema. Los objetos "ProxyServidor" y "CoordinadorDirAPO" junto con los demás que aparecen en la figura A1-9 se encapsulan en el componente "DirAPO_x" de la figura A1-4. Mientras que los objetos "ClienteDirAPO" y "ProxyCliente" se encapsulan en el componente "ClnteUsrBasico" (o "ClnteUsrAutorizado") de la figura A1-4. El diagrama de secuencias de interacciones de la figura A1-11 muestra la colaboración que se da entre los objetos de la figura A1-10.

[16] Se aclara que en DirAPO el sufijo "APO" alude a la frase "Abierto de Personas y Organizaciones", mientras que en el nombre de los componentes y objetos el prefijo APO alude a la frase "Agente para la Persistencia de Objetos".

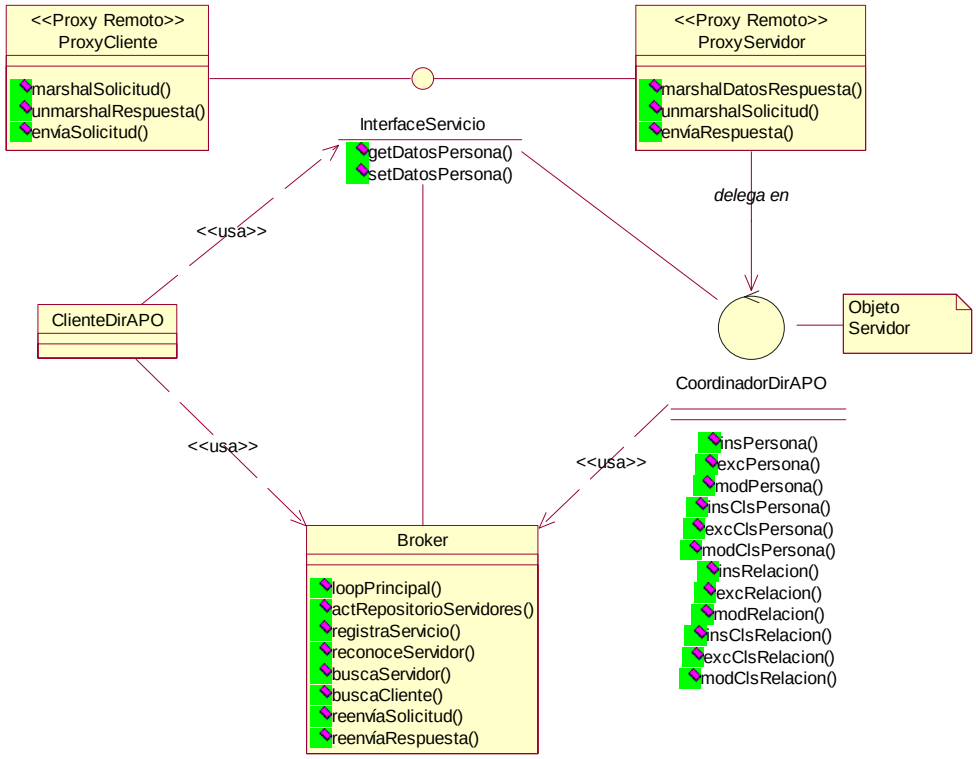

Figura #A1-10: Aplicación de “Proxy” de [GoF] para desacoplar ‘brokers’ y clientes según recomendación de [POSA1].

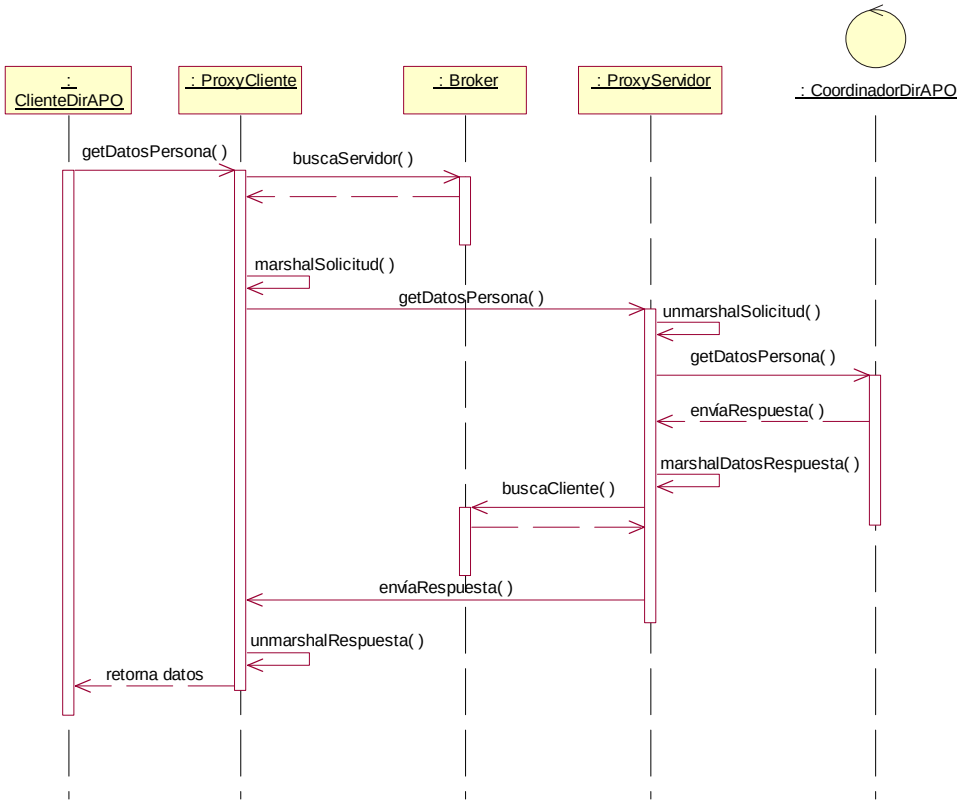

Figura #A1-11: Aplicación de "Proxy" de [GoF] para desacoplar "clientes" y "servidores" al usar "Broker" como partrón arquitectónico.

El patrón "Reflexión" de [POSA1] se categoriza como patrón arquitectónico pero en el contexto del sistema DirAPO sólo se puede apreciar su aplicación desde la perspectiva del diseño detallado de clases de análisis. "El patrón arquitectónico Reflexión provee un mecanismo para cambiar la estructura y comportamiento de los sistemas de software en forma dinámica. Permite la modificación de aspectos fundamentales tales como tipos de estructuras y mecanismos de invocación de funciones. En este patrón, una aplicación se divide en dos partes. Un nivel 'meta' provee información acerca de las propiedades del sistema y hace que el sistema sea 'autoconsciente'. Un nivel base incluye la lógica de la aplicación. Su implantación se construye sobre el nivel 'meta'. Los cambios a la información representada en el nivel 'meta' afectan el comportamiento subsecuente del nivel base." (pág. 193 de [POSA1]).

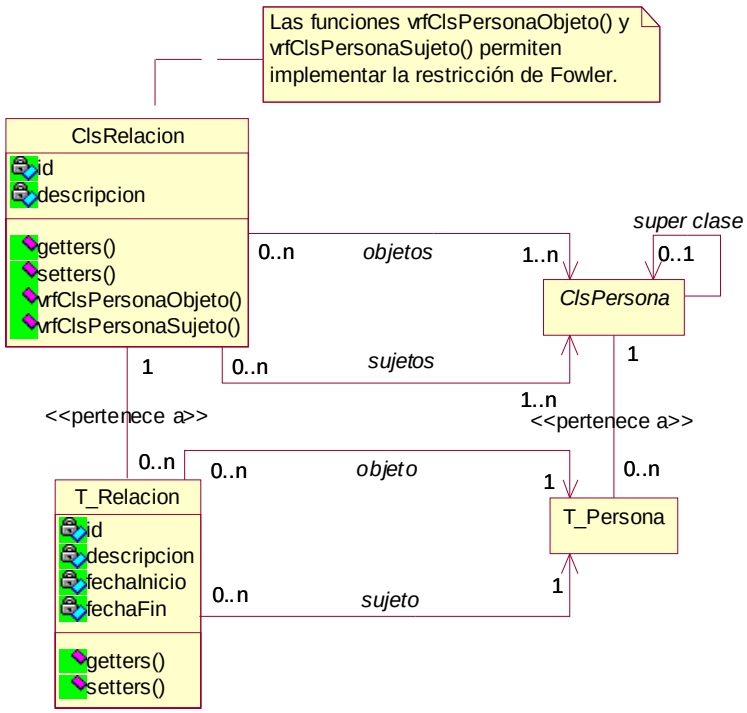

Figura #A1-12: Aplicación de "Reflexión" de [POSA1] implícita en "Accountability" de [Fowler 1997].

"Reflexión" está implícito en "Accountability" de [Fowler 1997] (véase la figura #A1-1. El nivel 'meta' aludido en la cita anterior abarca las clases "ClsRelacion" y "ClsPersona" presentadas en la figura #A1-12, así como sus conexiones. "ClsRelacion" permite que el conjunto de relaciones posible entre empresas se modifique dinámicamente, conforme sea necesario. Las conexiones entre "ClsRelacion" y "ClsPersona" permiten que el comportamiento del nivel base del sistema, conformado por las T_Relacion y T_Persona y sus conexiones, se modifique; por ejemplo impidiendo (o permitiendo) cierta clase de relación entre dos organizaciones previo registro de las "meta-conexiones" correspondientes, pues ese nuevo "conocimiento" del nivel 'meta' del sistema es aplicado cada vez que se intenta establecer una relación entre dos personas a través de la invocación de las funciones "vrfClsPersonaObjeto()" y "vrfClsPersonaSujeto()". La jerarquía de clasificación de personas también se define dinámicamente, no es un parámetro con el que se configura el sistema, sino que se adapta a las cambiantes necesidades de la empresa.

En resumen, se puede apreciar que los beneficios directos de introducir patrones de diseño detallado en la construcción del modelo de clases de análisis ha permitido concretar niveles apropiados de flexibilidad, adaptabilidad y reutilizabilidad del sistema. Los esquemas de solución de estos patrones de diseño probablemente ya han sido aplicados por el lector en sus tareas de análisis y diseño de sistemas. En tal caso, el beneficio de conocer los patrones podría ser analizar tales esquemas desde una perspectiva más general, disociada de los contextos específicos en que se los "descubrió", lo cual constituye una reflexión cotidiana sobre el quehacer profesional nada despreciable, si lo que se busca es mejorar continuamente las destrezas técnicas.

5. Conclusiones del apéndice

Concluida la elaboración del modelo conceptual de objetos, el diseño arquitectónico y el diseño detallado de objetos del sistema DirAPO cabe enfatizar que:

1. Los patrones se complementan entre sí. Se ha mostrado cómo “Fachada” y “Objeto de Valor” complementan la aplicación de “Multicapas”. Igualmente, se mostrado como “Proxy” complementa la aplicación de “Broker”.
2. El uso de algunos patrones es casi ineludible y múltiple. Este es el caso de “Composición”. Casi en cualquier sistema surgirá la necesidad de construir una representación flexible de una jerarquía de composición de objetos que además requiera un tratamiento uniforme de los objetos simples y los compuestos.
3. La utilización de patrones provee características al diseño detallado de un sistema que en general son deseables como flexibilidad, adaptabilidad y reutilizabilidad.

Apéndice 2: Recursos en la Web sobre patrones

La siguiente tabla muestra algunos sitios en la Web que proveen información valiosa sobre patrones. Todos han sido visitados por última vez el día primero de febrero del 2006.

Nombre del Sitio	**Dirección**	**Descripción**
Patterns Home Page	http://hillside.net/patterns/Pattern	Sitio principal sobre patrones en Internet. Provee conexiones a sitios de conferencias. Provee acceso a un repositorio de patrones.
Pattern Languages of Programming Conference	http://st-www.cs.uiuc.edu/~plop/	Antiguo sitio de la Conferencia de U de Illinois. El nuevo sitio aparece referenciado en el sitio de la fila anterior.
Software Engineering with Analysis Patterns	http://wwwai.wuwien.ac.at/~hahsler/research/virlib_working2001/virlib/virlib.html	Sitio de Hashler sobre patrones de análisis y su utilidad en la ingeniería del sofware
Martin Fowler	http://martinfowler.com	Sitio personal de Fowler que permite el acceso a documentos relevantes de este autor.
Joe's Pattern Reference	http://www.joeyoder.com/papers/patterns/patterns.html	Sitio personal de Joseph Yoder que permite el acceso a documentos relevantes de este autor.
Portland Pattern Repository	http://c2.com/ppr/index.html	Sitio del repositorio de patrones de Portland. Incluye referencias
About the Portland Form.	http://c2.com/ppr/about/portland.html	Describe el esquema para documentar patrones de Portland.
Interaction Design Patterns Page	http://www.visi.com/~snowfall/InteractionPatterns.html	Sitio con documentos importantes sobre patrones de interacción humano-sistema.
Patterns of Interaction: a Pattern Language for CSCW	http://www.comp.lancs.ac.uk/computing/research/cseg/projects/pointer/pointer.html.	Sitio con documentos importantes sobre patrones de interacción humano-sistema.
Enterprise Solution Patterns Using Microsoft .NET	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/Esp.asp	Sitio de la MSDN de Microsoft que describe e ilustra el uso de patrones en su plataforma de desarrollo .Net.
Core J2EE Patterns: Patterns index page	http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html	Sitio de la empresa Sun Microsystems especializado en el uso de patrones en su plataforma J2EE.