La Geomática y la factoría inteligente: integración de "software" utilizando Inteligencia Artificial y Orientación a Objetos

Dr. José A. Navarro Esteban

Institut Cartogràfic de Catalunya
Parc de Montjuïc
08038 Barcelona
pep@icc.es

Como en otras disciplinas, el proceso de datos es complejo y heterogéneo en Geomática. La constante aparición de nuevos sensores, productos, formatos de datos o algoritmos etc., hace que la construcción de "software" específico sea un problema que se agrava cuando el proceso es masivo y es necesario producir de forma industrial grandes volúmenes de información. En esta situación, es casi obligatorio disponer de sistemas tipo "factoría", los cuales integran multitud de aplicaciones. El diseño de estas factorías no es un proceso sencillo; al contrario, es complejo y requiere un conocimiento profundo tanto de las características del producto buscado como de las de las herramientas "software" a integrar y del correcto flujo de información entre éstas. Aquí se presenta una solución para aliviar este problema que se apoya en técnicas de inteligencia artificial para formalizar el conocimiento de los expertos de dominio y aprovecharlo de forma sistemática para diseñar de forma automática flujos de producción apoyados en herramientas "software" diseñadas según los principios de Orientación a Objetos.

1. Introducción

El proceso industrial (masivo) de datos es muy importante dentro del ámbito de la Geomática. Si bien es más que habitual encontrar situaciones en las cuales el objetivo final de un proyecto consiste en crear un número reducido de unidades de un producto, es bastante frecuente afrontar el caso diametralmente opuesto, donde es posible llegar a contar por centenas o incluso millares el volumen de elementos producidos. Un caso típico, por ejemplo, sería una serie ortofotográfica a escala 1:5.000 cubriendo el área de una comunidad autónoma.

El tipo de herramientas "software" utilizadas en este último tipo de proyectos condiciona enormemente la productividad que puede llegar a conseguirse. Evidentemente, cuanto mayor sea el grado de automatización e integración de las diferentes aplicaciones necesarias, mayor será el rendimiento obtenido. Es conveniente, por lo tanto, disponer de "software" orientado a la producción de proyectos completos en lugar de sistemas cuyo objetivo sea la elaboración de unidades aisladas.

Esto puede verse aún más claramente si se tiene en cuenta que normalmente es preciso utilizar una pléyade de aplicaciones para conseguir viajar desde los datos iniciales (sean estos del tipo que sean) al producto acabado. La utilización individualizada –y posiblemente, manual- de dichas herramientas para resolver cada una de las etapas de las cuales consta el proceso de elaboración puede penalizar muy gravemente el tiempo necesario para completar un proyecto.

Por estas razones, y cuando de producción masiva se trata, es muy importante disponer de sistemas "software" que emulen el funcionamiento de una factoría clásica. Desde luego, para poder aplicar este tipo de proceso, es necesario que los condicionantes de producción sean los de fabricación en masa: el tipo de proceso es idéntico para todos los ítems –si bien, en el caso de factorías "software", la materia prima, los datos, es diferente para cada unidad producida. Las "factorías software" no tienen sentido cuando cada unidad de producto exhibe unas características diferenciadas. En este caso, y siguiendo con la analogía, podría hablarse de producción artesanal, pero este no es el caso que se discute en este artículo.

Existen otros condicionantes que determinan la posibilidad de emplear factorías de "software" y son de tipo más tecnológico. Pero antes de comentarlos, conviene definir que es una factoría de "software" dentro del contexto de este artículo.

> *Una factoría de software es un sistema informático integrando un número determinado de* aplicaciones informáticas *(programas), cada una de las cuales realiza una función determinada dentro de una cierta disciplina de conocimiento así como un conjunto de* flujos de datos *entre las citadas aplicaciones, los cuales determinan el* orden

en que aquellas han de ejecutarse. La utilización de este sistema conduce invariablemente a la creación del mismo tipo de producto.

La Figura 1 ejemplifica la definición anterior.

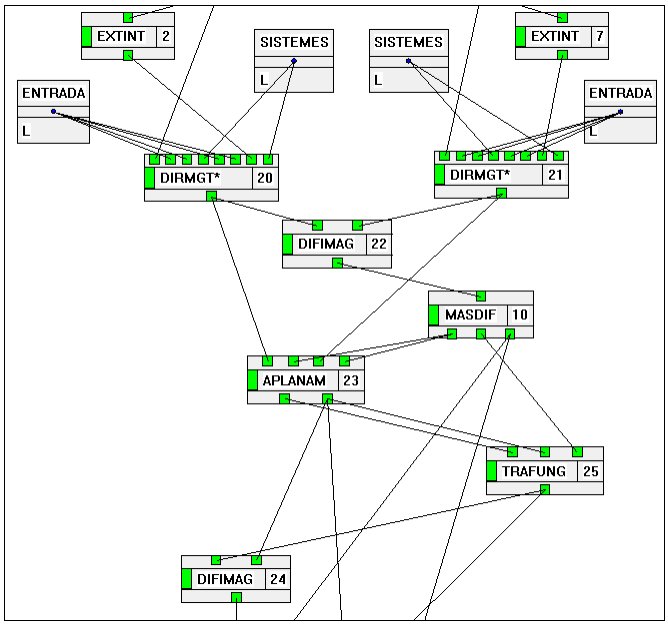

Figura 1: Fragmento de una factoría

En esta figura, las aplicaciones informáticas o programas están representados por rectángulos con pequeñas cajas en la parte superior e inferior. Estas, a su vez, representan los archivos de entrada (arriba) o de salida (abajo) utilizados por el programa en cuestión. El flujo de datos queda materializado por las líneas conectando los programas. El orden del proceso queda implícito. Además, aparece un elemento adicional, los repositorios de datos (los otros rectángulos, como el que se identifica con la etiqueta ENTRADA) que representan lugares desde donde se toma o deposita la información necesaria para la ejecución de la factoría.

Desgraciadamente, incluso cuando se trata de producir grandes volúmenes de datos, no siempre es posible utilizar esta aproximación. Las factibilidad del concepto factoría de "software" depende, entre otros factores, de la capacidad de *integrar* los diferentes programas de que se compone, de manera que su ejecución pueda ser *desasistida* -automática. Si esos programas no están construidos siguiendo unas determinadas pautas normalizadas, será totalmente imposible incluirlos en una factoría. Por ejemplo, una aplicación con una interfaz gráfica de usuario difícilmente podrá ser utilizada de este modo; por el contrario, el clásico programa de línea de comandos podrá ser incorporado mucho más fácilmente.

Por lo tanto, y debido a estos condicionantes puramente tecnológicos, casi podría afirmarse que el concepto de factoría de software solamente puede utilizarse cuando el desarrollo de los programas involucrados es propio y no de terceros, a no ser que estos adopten la normativa de desarrollo en cuestión.

En el caso del Institut Cartogràfic de Catalunya (ICC) se satisfacen los condicionantes necesarios para hacer posible este tipo de proceso. Tanto la necesidad de producción masiva como la existencia de un fondo de aplicaciones desarrolladas internamente, permiten utilizar este concepto de forma muy satisfactoria. En la Sección 2 se describe brevemente un sistema en esta línea totalmente operativo.

2. Precedentes: ORTO y Visual Factory Suite

El ICC está aplicando el paradigma de factoría "software" desde 1.988. En aquel entonces, se diseñó el sistema ORTO (Ortofotos Rectificadas por Trozos en modo Offline, véase [1]), todavía operativo, con la intención de producir de forma industrial la ortofoto de Cataluña a escala 1:5.000 en blanco y negro. Desde aquel momento, el sistema ha sido mejorado en

sucesivas ocasiones y puede afirmarse que la gran mayoría de los proyectos de ortofotografía del ICC se han realizado utilizando este sistema.

ORTO, a pesar de sus muchas limitaciones (es un concepto de hace ya doce años) ya incorpora la filosofía de ejecución desasistida y el objetivo de un alto rendimiento. Una vez definida una factoría, ejecuta de forma automática todos los programas que la componen en el orden correcto, gestionando asimismo las transferencias de datos desde / a los repositorios de información establecidos. No obstante, este sistema adolece de una serie de defectos que le impiden hacer frente con elegancia a dos problemas que se dan habitualmente en el proceso de datos dentro de cualquier disciplina: el *cambio* y el *ensamblado*.

Como se apuntaba en el inicio de este trabajo, no solamente dentro del ámbito de la Geomática sinó de muchas otras disciplinas, el cambio es constante: aparecen nuevas fuentes de datos, algoritmos mejorados, necesidad de obtener nuevos productos o de utilizar nuevos formatos de datos, etc. Evidentemente, esto tiene un impacto directo en las factorías "software": es necesario adaptar las ya existentes o crear otras nuevas para afrontar los nuevos requerimientos. Y esto ha de hacerse de forma rápida para conseguir que el tiempo transcurrido entre la aparición de la necesidad y su realización sea el mínimo posible.

Por otra parte, una factoría está compuesta por un conjunto de programas que han de ejecutarse en un cierto orden, usando unos determinados archivos. Es la ejecución de estas piezas independientes lo que nos permite elaborar el producto. Evidentemente, es preciso orquestar esa ejecución para que los archivos utilizados sean los correctos, para que los programas se invoquen de la manera adecuada y en el orden apropiado, etc. En otras palabras, es necesario ensamblar las piezas de forma conveniente y, sobre todo, automatizada, evitando cualquier intervención manual. Este ensamblado es responsabilidad del sistema de gestión de factorías "software".

En el caso de ORTO los dos problemas anteriores están resueltos, sí, pero no de forma eficaz. Tanto la forma en que se define una factoría como los pasos necesarios para efectuar el ensamblado de sus componentes (programas) hacen precisa la intervención del equipo de desarrollo de "software", el cual tiene que poseer unos conocimientos bastante profundos del sistema para poder realizar estas tareas correctamente. Eso implica dos cosas: necesidad de personal muy cualificado y tiempos de entrega de la factoría solicitada bastante elevados (15 días en media). Además, y como corolario, el equipo de explotación, los auténticos expertos en la producción de ortofotos, no pueden diseñar sus factorías por si mismos, creándose una fuerte dependencia respecto del equipo de desarrollo.

A pesar de todo, la experiencia ORTO ha resultado no solo positiva, sino muy satisfactoria; además, sentó las bases y creó la experiencia para finalizar la construcción, en 1999, del nuevo sistema de diseño y explotación de factorías del ICC, la Visual Factory Suite (VFS, véase [5]). Mediante este conjunto de herramientas, se resuelven completamente los problemas detectados durante la explotación de ORTO.

Para ello, la VFS se articula según el siguiente "ciclo de vida":

- Diseño de la factoria.
- Planificación de los detalles reales de explotación.
- Solicitud de producción.
- Monitorización y control del sistema.

Es muy importante destacar que la VFS no está ligada en absoluto a ninguna disciplina en particular. Por el contrario, es independiente del campo de aplicación. Si bien la producción de ortofoto digital es el primer lugar donde el ICC utilizará esta nueva herramienta, VFS es completamente general.

Las tareas realizadas en los diferentes pasos del ciclo de vida introducido arriba son las siguientes:

Diseño de la factoría. Mediante una herramienta gráfica, el Visual Factory Builder (VFB), el ingeniero de producción combina los diferentes programas disponibles en el modo y orden convenientes. Este modo de operación es muy intuitivo, y traslada absolutamente al equipo de producción las responsabilidades de diseñar *como* se genera un producto. Además, VFB no permite realizar diseños incorrectos desde el punto de vista sintáctico (como por ejemplo, alimentar un programa con un archivo cuyo tipo no sea el adecuado) con lo cual el ingeniero de producción solo tiene que concentrarse en los aspectos relacionados con el modo en que se tiene que producir. Es importante destacar que a partir del momento que una factoría ha sido diseñada, ésta está *prácticamente lista para ser ejecutada*. "Diseñar y explotar", éste es el slogan. La Figura 2 muestra VFB la edición de una factoría en curso.

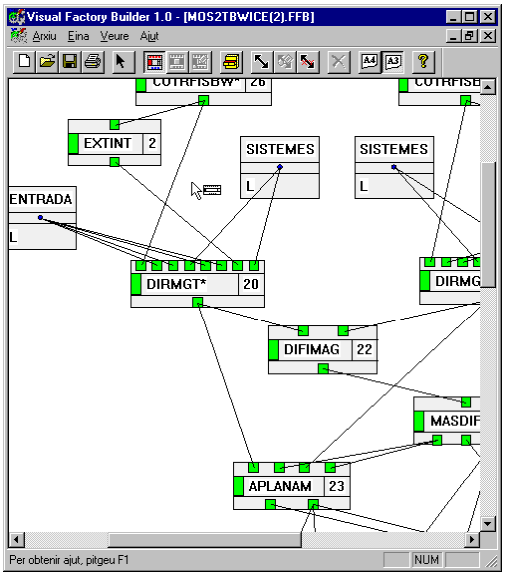

Figura 2: Visual Factory Builder

Planificación de la explotación. Cuando se diseña una factoría, no se fijan detalles físicos de explotación. Por ejemplo, los repositorios de datos no se ubican en directorios físicos, no se establecen cuotas sobre estos o no se indica que ordenador (u ordenadores) serán responsables de llevar a cabo la ejecución. Mediante Factory Planner, otra herramienta de la VFS, el responsable de la explotación establece, entre otros, todos estos parámetros. La gestión se realiza en base a diferentes proyectos, de manera que una misma factoría puede ser utilizada en más de uno de ellos, adecuándola en cada caso a las necesidades particulares existentes. Esta personalización va todavía más lejos.

Los programas que integran una factoría trabajan con dos tipos de información: aquella que se encuentra depositada en archivos y la que se suministra vía teclado (parámetros de la ejecución). Durante la explotación de un proyecto, muchos de estos parámetros se fijarán para todos los ítems producidos; por tanto, no debería molestarse al operador interrogándolo sobre su valor cada vez que se procesa uno de ellos. Factory Planner permite realizar esta preasignación, siempre según cada proyecto.

No será extraño encontrarse con parámetros "repetidos" entre diferentes programas de la factoría. Por ejemplo, si ésta trabaja en con ortofotos, el tamaño del "píxel" de la imagen resultante será un parámetro que probablemente habrá que suministrar a más de uno de los programas que la integran. Evidentemente, si no se toman medidas adicionales, el operador tendrá que contestar a la misma pregunta más de una vez con el mismo valor como respuesta. Con Factory Planner es posible agrupar todas las preguntas cuya contestación haya de ser la misma de manera que el operador solo sea interrogado una vez.

Es importante insistir en el hecho de que toda esta planificación se realiza en base a uno u otro proyecto, de manera que es posible utilizar la misma factoría personalizándola según las necesidades de cada uno de ellos.

Y por último: una vez realizada la planificación, la factoría está lista para ser explotada.: ¡sin intervención del equipo de desarrollo!

La Figura 3 muestra el panel principal de Factory Planner, donde puede apreciarse la jerarquización por proyectos de las factorías, así como los ordenadores disponibles para su ejecución. En la Figura 4 puede apreciarse la preasignación y agrupación de parámetros de teclado.

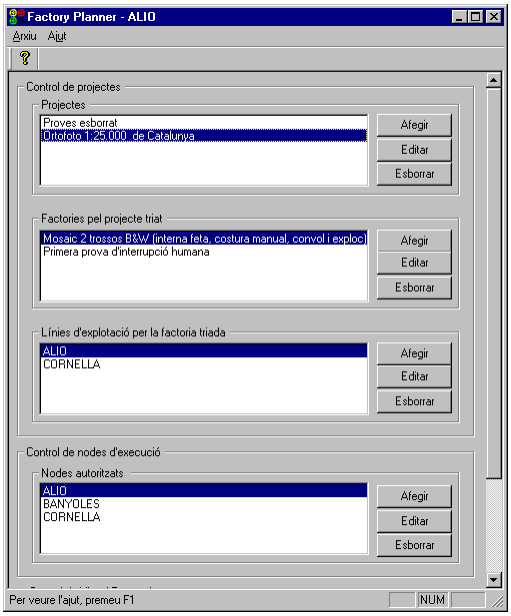

Figura 3: Panel principal de Factory Planner

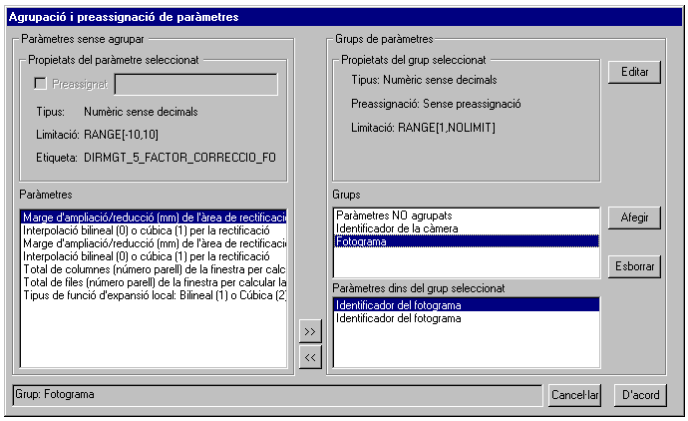

Figura 4: asignación y agrupación de parámetros en Factory Planner

Explotación. En este momento el operador puede iniciar la producción de ítems basándose en la factoría ya planificada. Para ello, utiliza la herramienta Factory Launcher, la cual le permite seleccionar, entre aquellos disponibles, el proyecto y la factoría de su interés. Una vez hecho esto, es necesario dar valores a los parámetros de ejecución que no se han preasignado en la fase anterior. A partir de este punto, puede ponerse en marcha la factoría.

La herramienta Factory Manager, que ha de estar permanentemente en marcha, recibe el encargo de producción y, una vez identificados convenientemente el proyecto, la factoría y los parámetros introducidos, se encarga de ejecutar los programas involucrados en el orden correcto y usando los archivos necesarios.

La Figura 5 muestra el aspecto de Factory Launcher a punto de iniciar el proceso de un ítem. Obsérvese la jerarquización en proyectos / factorías y ordenadores idéntica a la de Factory Planner. Además, aparece la lista de preguntas a contestar –y en particular, puede verse la respuesta a la última de ellas- por el operador.

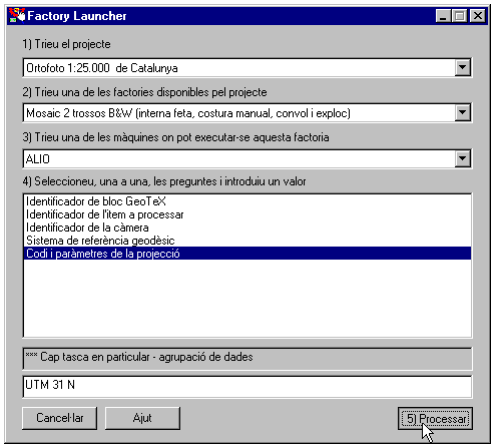

Figura 5: Factory Launcher

Monitorización y control. Evidentemente, una vez iniciado el proceso de uno o más ítems, es necesario conocer el estado global del sistema y ser capaces de controlarlo en respuesta a las diferentes vicisitudes que puedan darse. Para ello, la VFS incluye la herramienta Visual Factory Tracker (VFT). Mediante VFT es posible mostrar, de forma gráfica, el estado de todos los ítems en proceso. Esto incluye detalles sobre proyectos, factorías, ordenadores, tiempos de proceso e, incluso, el programa exacto (dentro de la factoría) que se está ejecutando para cada ítem dentro del sistema. Adicionalmente, como ya se ha apuntado, es posible interaccionar con el sistema. Ejemplos de estas interacciones son: control de prioridad de ejecución entre ítems, eliminación de estos, o parada controlada entre otras. Finalmente, VFT se comporta también como un centro de recepción de alarmas que puedan producirse durante la explotación; el responsable de operación las recibe y puede así tomar las medidas oportunas para subsanarlas.

La Figura 6 muestra Visual Factory Tracker en acción.

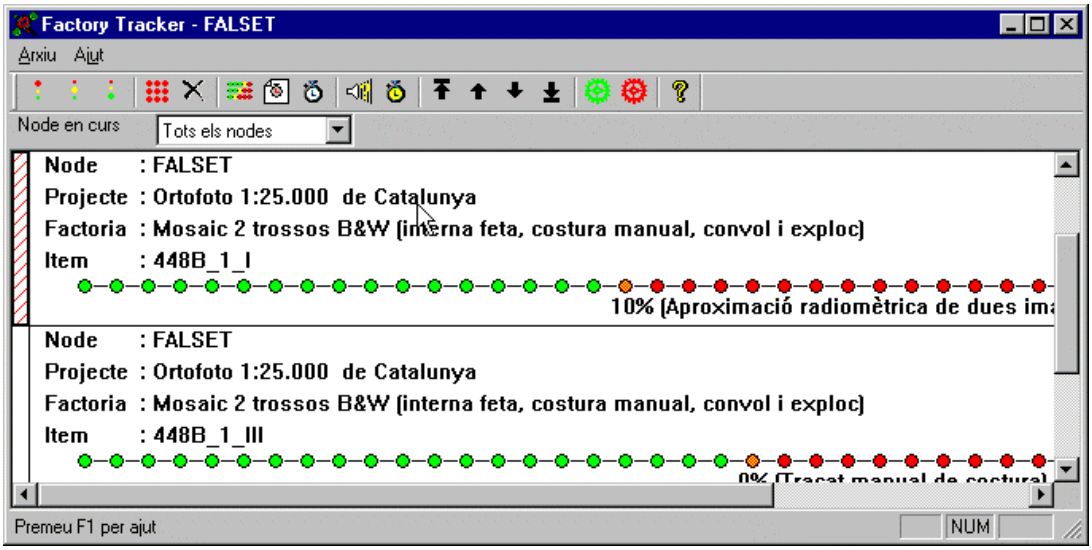

Figura 6: Visual Factory Tracker

Finalmente, y aunque no forme parte del ciclo de vida de la VFS, hay que destacar que ésta es capaz de gestionar de forma automática tareas que tengan que realizarse por humanos. Este tipo de situaciones se da típicamente cuando no existe un programa capaz de realizar una determinada tarea de forma completamente desasistida –por ejemplo, porque no existe un algoritmo "seguro" para realizarla. En estos casos, es necesario que un experto realice manualmente esa tarea.

Si no se toman medidas adicionales, las tareas "manuales" rompen el flujo de producción por automático, ya que es necesario interrumpir éste para que el experto realice el proceso. Para evitarlo, es posible introducir "interrupciones humanas" dentro del editor de factorías (Visual Factory Builder). Estas interrupciones se tratan como un programa más en todos los aspectos, con una diferencia básica: cuando durante el proceso de explotación se llega a la citada interrupción, se notifica al responsable de explotación que se ha llegado a ese punto y que es preciso que "alguien" suministre los archivos resultantes de la tarea manual. Cuando esto se consigue, se notifica al sistema de que puede continuar y el proceso se reinicia automáticamente.

Por medio de esta técnica no es posible evitar la necesidad de intervención manual, pero la gestión del flujo de producción automatizado es extremadamente sencilla.

3. La factoría inteligente

La Visual Factory Suite representa un gran salto respecto a ORTO en tanto que resuelve dos problemas graves que este último sistema no trataba con la suficiente flexibilidad y eficacia: el cambio y el ensamblado. No obstante, existe otro problema que la VFS no puede paliar completamente: el de la *complejidad*.

Una factoría puede llegar a ser extremadamente complicada. A pesar de que una herramienta como Visual Factory Builder, el editor de factorías, simplifique enormemente el proceso de diseño de éstas evitando que el ingeniero de producción tenga que preocuparse de detalles técnicos, no puede ayudarlo diciéndole *como* tiene que combinar los programas para llegar al producto deseado.

Esto puede parecer una perogrullada, pero no es así. Cuando se trata de diseñar factorías de la "vida real", el número de programas involucrados puede ascender con facilidad a varias decenas. Si se tiene en cuenta, además, que una misma tarea puede ser resuelta por medio de diferentes programas o combinaciones de estos, el argumento se ve más reforzado.

El problema de la complejidad no puede atacarse con los mismos recursos que con los de complejidad y el ensamblado, ya que en este caso el obstáculo principal a superar es el de la necesidad de inteligencia humana. Por lo tanto, resolver, o al menos paliar este problema requiere mecanismos que permitan emularla. Pero antes de continuar con esta discusión, sería interesante clarificar cual es el objetivo que se oculta tras la expresión "factoría inteligente" que da título a esta sección.

Un sistema como la Visual Factory Suite permite, de forma muy cómoda y eficaz, diseñar y explotar factorías, recayendo toda la responsabilidad –excepto la de la construcción de los programas que se integran en las factorías- en las manos del personal realmente experto en producción. Pero para poder funcionar requiere que la fase de diseño sea realizada por un experto del dominio en cuestión; a partir de este punto, todo es automatizable. En el caso de una factoría inteligente, la tarea de diseño debería ser también automática.

Así pues, el objetivo planteado –o al menos, el primero de ellos- es la de ser capaces de *diseñar, automáticamente, factorías*. Una factoría así creada es la que denominaremos "inteligente". Y será inteligente porque durante su proceso de diseño se habrá utilizado todo el saber hacer de los ingenieros de producción, que son quienes en realidad conocen como deben resolverse los problemas dentro de la disciplina en la que trabajan. A un sistema como la Visual Factory Suite solamente le falta un diseñador de factorías inteligente para trabajar de este modo. El resto de este trabajo describe como es posible conseguirlo. No obstante, se da una visión bastante general de los conceptos involucrados y de los resultados obtenidos, ya que en [4] puede encontrarse una descripción exhaustiva.

4. Una aproximación general,: el "framework" inteligente

En lugar de enfocar de forma particular el modo de conseguir un diseñador inteligente de factorías que se adecue a las necesidades particulares de la Visual Factory Suite, se adoptará una aproximación más general al problema, ya que de esta manera los resultados obtenidos serán aplicables a otros niveles.

Diseñar una factoría no deja de ser un proceso de generación de "software". En otras palabras, lo que se acaba obteniendo es un "programa" mucho mayor que sus componentes, resultante de la integración de estos, cuya función es la de obtener el producto para el cual ha sido concebido.

Si se observa un programa bajo la óptica del párrafo anterior, podrá observarse que la definición dada es igualmente válida para éste. Alterando algunos de los sustantivos empleados, puede decirse que un programa es el resultado de la combinación de una serie de llamadas a subrutinas (o la invocación de métodos de unas ciertas clases dentro del ámbito de la orientación a objetos), cuya función es la de obtener el producto para el cual ha sido concebido. Y puede irse más lejos, llevando esta

definición al terreno de una subrutina (o método). Por lo tanto, diseñar una factoría de forma inteligente no deja de ser lo mismo, al menos desde el punto de vista conceptual, que generar un programa, rutina, o método de una clase utilizando inteligencia.

Como puede verse, el concepto es aplicable a diversos niveles de la ingeniería de "software". Obteniendo un generador de software inteligente que sea capaz de tratar, por ejemplo, con subrutinas o métodos en una clase, los resultados serán extrapolables al resto de los niveles mencionados. Por esta razón, será el "framework", la máxima expresión de la tecnología de orientación a objetos, el punto de partida desde el cual se desarrollará este trabajo.

Un "framework", en el sentido clásico de la palabra, se compone de un conjunto de clases interrelacionadas entre si y que son capaces de solucionar familias de problemas parecidos, evidentemente dentro de un determinado campo de aplicación (como por ejemplo, ingeniería eléctrica o transformación de coordenadas). Resolver un problema dentro de un "framework" implica, como de costumbre, combinar una serie de llamadas a métodos de las clases disponibles en un cierto orden.

Pero la orientación a objetos no es suficiente para poder a cabo la tarea impuesta. Se ha comentado anteriormente que para poder hacer frente al problema de la complejidad es necesario inteligencia. Concretamente, la de los expertos del dominio de trabajo. Por lo tanto, es vital introducir ese conocimiento dentro del "framework" para poder convertirlo en inteligente.

Tradicionalmente, este tipo de problemas se han solucionado a través de sistemas expertos (SE). Existen múltiples ejemplos de aplicaciones basadas en esta tecnología –las pioneras, especialmente en el campo de la medicina. Para poder emplear un SE es imprescindible *formalizar* el conocimiento, el saber hacer de los expertos del ramo. En otras palabras, se trata de plasmar de forma ordenada todo el "cuerpo de doctrina" que se está usando para hacer frente a una serie de problemas. Ese conocimiento puede provenir de orígenes tan dispersos como la intuición, algoritmos probados, análisis estadísticos o simples reglas heurísticas. Los SEs, además, incorporan motores de inferencia capaces de imitar el modo en que piensan los humanos. Son, por lo tanto, capaces de extraer conclusiones, de "razonar".

Por lo tanto, el sistema experto será la herramienta utilizada para complementar el "framework" clásico, para dotarlo de inteligencia. Éste, aportará los algoritmos existentes, las piezas a partir de las cuales se construirán las soluciones a problemas de mayor complejidad; el SE contribuirá con un mecanismo de razonamiento para deducirlas, empleando el saber hacer humano.

Ya han aparecido dos de los componentes esenciales. Todavía falta un tercero, de gran importancia. Se trata del conocimiento sobre los algoritmos (métodos) disponibles en las clases incluidas en el "framework". Esto es así porque para poder combinar esos métodos es imprescindible conocer su semántica: qué es lo que hacen y con que tipo de objetos trabajan. Este conocimiento, nuevamente ha de ser formalizado para poder ser utilizado sistemáticamente.

Así pues, los tres pilares básicos del la factoría inteligente son, como acaba de apuntarse,

- El "framework" clásico, incluyendo las clases con los algoritmos disponibles,
- la formalización de la semántica de los métodos contenidos en esas clases y
- el sistema experto, que contiene el conocimiento sobre el dominio de aplicación y un motor de inferencias para extraer conclusiones.

Falta todavía identificar al responsable de combinar todos esos recursos para ser capaz de diseñar nuevas factorías. Puesto que se parte del "framework" como lugar donde se encuentra depositada toda la algorítmica relacionada con el dominio, el lugar apropiado para ubicarlo es el propio "framework". Y desde el momento en que se trabaja con tecnología orientada a objetos, será una nueva clase dentro de aquel quien tendrá que asumir esas responsabilidades. En particular, será la *clase inteligente*.

Si bien todos los elementos necesarios para construir un framework inteligente, y por tanto, la factoría inteligente, ya han aparecido, es necesario incluir aún un componente adicional. Se trata del *protocolo de conocimiento*. La razón por la cual es necesario proceder así es la de la *normalización*.

Una clase inteligente tiene que ser capaz de encontrar soluciones independientemente del dominio de conocimiento dentro del cual se esté trabajando; eso quiere decir que el *tipo de uso* que se haga de dicho conocimiento ha de ceñirse a unas determinadas reglas. Por otra parte, el conocimiento depende de la disciplina con la que se está trabajando. Esto, aparentemente, es una contradicción, pero es resoluble.

Si bien no puede hacerse ninguna suposición sobre el contenido del conocimiento almacenado en el sistema experto, es posible organizarlo de la manera que le conviene a la clase inteligente. Como se verá en la Sección 5, donde se describe detalladamente como se orquestan todos los elementos introducidos hasta ahora, esto permite efectuar esa normalización que garantiza la independencia necesaria.

El protocolo de conocimiento, pues, no es más que una forma estandarizada que ha de respetar la clase inteligente para acceder a la información gestionada por el sistema experto.

La Figura 7 muestra la arquitectura global del framework inteligente, con las interrelaciones existentes entre sus diferentes componentes. En la Sección 5 se discute su funcionamiento.

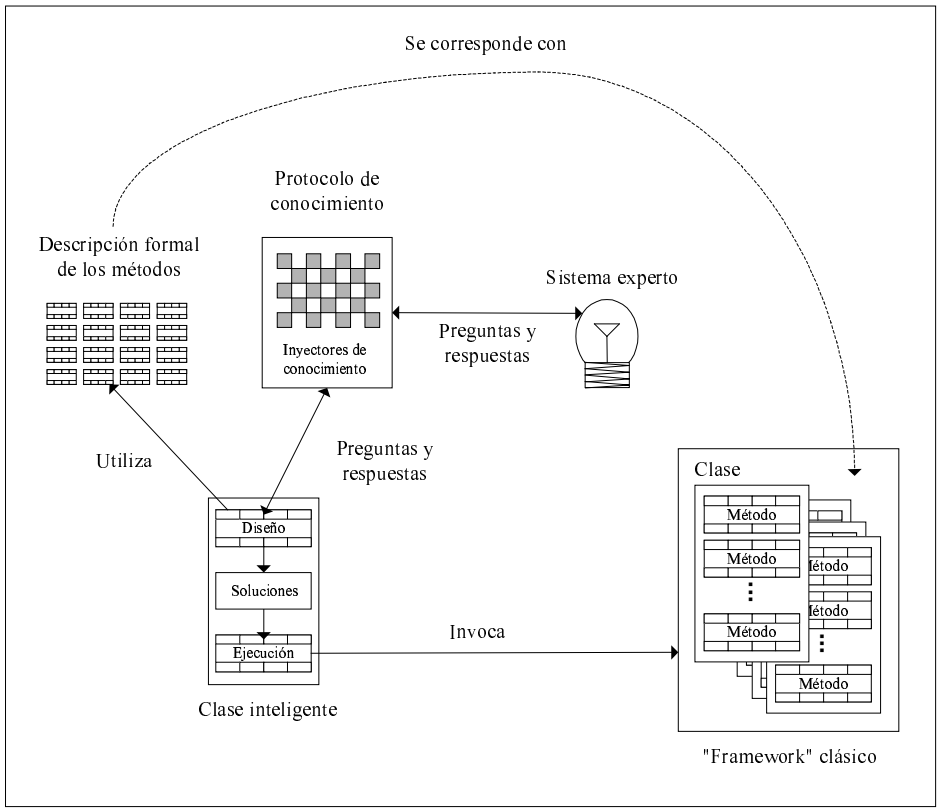

Figura 7: Arquitectura general de un "framework" inteligente

5. La clase inteligente

Así pues, la primera misión de la clase inteligente es la de encontrar el conjunto de soluciones (factorías) capaces de crear el producto buscado. La segunda, ejecutarlas.

Para ello, expone dos métodos diferentes: el "diseñador" y el "ejecutor" (véase la Figura 7). Si en lugar de pensar en un "framework" inteligente se traslada el contexto de la discusión a un sistema como la Visual Factory Suite, estos métodos corresponderían, respectivamente, a una versión inteligente de Visual Factory Builder en el primer caso y a Factory Manager en el segundo.

Si bien el método ejecutor es importante ya que su responsabilidad es la de poner en práctica las soluciones encontradas por el diseñador y obtener así el producto buscado, desde el punto de vista técnico no plantea demasiados problemas. Por esta razón, la discusión se centrará esencialmente en el diseñador.

El punto esencial de la discusión es *como* encuentra el diseñador esas soluciones. En general, la primera parte del procedimiento utilizado –que se basa en una óptica de consumidor / productor - es el siguiente:

- Se arranca de una especificación formal del producto requerido. En la Sección 6 se discute en que consiste esta formalización –que es válida tanto para describir el producto o problema como los métodos incluidos en el "framework" clásico.

- El diseñador identifica, gracias a la descripción formal de los métodos disponibles en el "framework" clásico, cuales de ellos son capaces de generar el producto solicitado. Pueden aparecer desde 0 hasta n métodos. Caso de que no haya ninguno, el proceso acaba, ya que no se dispone de ningún modo de generar el producto. Por el contrario, si aparece más de uno, existen múltiples soluciones.

- El proceso sigue recursivamente. Los métodos identificados en la etapa anterior tienen a su vez unas entradas que satisfacer. En otras palabras, es necesario generar o proveer esas entradas para poder utilizar los métodos obtenidos. Por lo tanto, para cada uno de esos objetos, se repite el proceso de busca de métodos capaces de generarlos.

- La búsqueda finaliza cuando no se encuentra ningún método para ninguno de los objetos pendientes de ser generados. Estos objetos no "solucionados" constituyen el conjunto de datos de entrada del problema.

La Figura 8 muestra una posible solución para un caso en que son necesarias tres iteraciones en total.

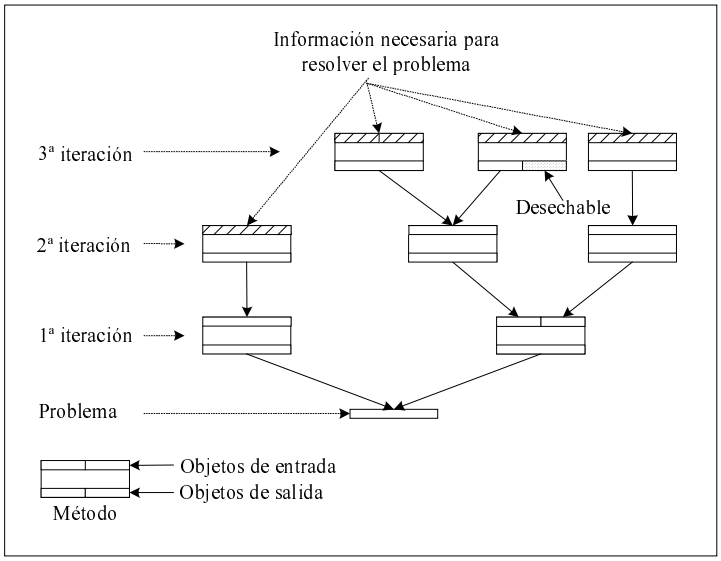

Figura 8: Primer paso en la búsqueda de soluciones; ejemplo en tres iteraciones con múltiples soluciones

Nótese que esta primera fase procede de una forma absolutamente mecánica y que en ningún momento se ha utilizado el conocimiento de los expertos para mejorar las soluciones. Ahora es el momento de hacerlo. Y para ello se utiliza el protocolo de conocimiento. No obstante, antes de describir como se utiliza ese saber hacer de los expertos humanos es necesario tener claro qué quiere decir "aplicar el conocimiento".

El conocimiento es necesario solamente cuando aparecen múltiples soluciones. Si no existe más que una forma de generar un producto, no hay opciones, no hay decisiones que tomar. En este caso, la única factoría diseñada por medio del proceso descrito arriba no precisa ser "mejorada", por no haber otros modos de conseguir lo que se busca.

Por el contrario, ese saber hacer puede ser explotado a fondo cuando hay múltiples posibilidades de proceso. En este caso, los objetivos a conseguir son purgar / ordenar (de mejor a peor, siempre según el saber hacer de los expertos) este árbol de soluciones. Para ello, se utilizan los siguientes procedimientos:

- *Eliminar métodos* que, bajo los condicionantes del problema a resolver no son aplicables. Por ejemplo, si se está realizando cartografía satélite, no tendrá sentido esforzarse en conseguir un modelo digital del terreno para rectificar una imagen, ya que el efecto que aquel tiene en el proceso es negligible.

- *Forzar la utilización exclusiva de uno de los métodos posibles*, descartando incondicionalmente el resto. Supóngase ahora que se está realizando ortofoto digital sobre núcleos urbanos; en este ejemplo, al contrario que en el anterior, no es solo importante utilizar un modelo digital del terreno, sino que además conviene que este sea de una calidad muy elevada (por ejemplo, organizado en triángulos en lugar de en malla). Por esta razón, solamente debería emplease aquel método de rectificación usando modelos digitales triangulares y rechazarse cualquier otro.

- *Ordenar según criterios de calidad* los métodos supervivientes si no se llega a una solución única. Los dos procedimientos anteriores están orientados a reducir el espacio de soluciones, ya sea eliminando métodos impracticables o fijando uno cuando las circunstancias así lo requieran. De todas formas, la aplicación de estos no siempre llevarán a conseguir una única solución para el problema dado. Es entonces cuando se aplica este tercer procedimiento para intentar establecer una ordenación de las soluciones restantes. Insistiendo en el ejemplo de ortofoto, podría llegar a establecerse un criterio que seleccionase en primer lugar un método u otro en función de la escala del producto.

Estas operaciones articulan la forma en que se ha de organizar el conocimiento dentro del sistema experto y la manera en que la clase inteligente accede a él. Y, de hecho, es lo que permite conseguir que la citada clase inteligente sea independiente del dominio al cual pertenece el problema a solucionar. Nótese que cada una de estas operaciones constituyen un punto de entrada (o "inyector de conocimiento") del protocolo de conocimiento.

En otras palabras, el saber hacer de los expertos ha de organizarse de tal manera que el sistema experto sea capaz de responder (en este caso) a las tres preguntas que se deducen de las operaciones descritas:

- Debería eliminarse el método X dados los condicionantes de este problema?
- Existe algún método Y que debería utilizarse incondicionalmente en este caso? Y
- Es mejor el método Z que el método T en la situación planteada por este problema?

Estos tres "inyectores", en el prototipo de clase inteligente que se ha implementado para probar este concepto de diseñador inteligente, se han llamado "never", "always" y "better" (nunca, siempre y mejor que respectivamente). Y son los únicos utilizados por éste. Evidentemente, puede pensarse en añadir otros inyectores, enriqueciéndose así la forma en que puede organizarse el conocimiento en el sistema experto y ser utilizada por la clase inteligente a través de su método diseñador. Por ejemplo, podría pensarse en un inyector denominado "por contrato". Este inyector podría servir para indicar que un determinado método ha de utilizarse en lugar de cualquier otro porque el cliente así lo exige, independientemente de consideraciones de calidad del producto que puedan plantearse.

La Figura 9 muestra como podría quedar el árbol de soluciones encontradas anteriormente (ver la Figura 8). En este caso, se ha ejemplificado la desaparición de un método –y de una solución como consecuencia- y la ordenación de las soluciones resultantes.

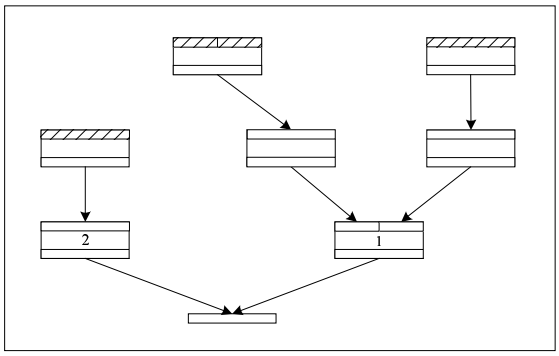

Figura 9: Soluciones purgadas y ordenadas según el conocimiento de los expertos

Llegando a este punto, la tarea de método diseñador es puramente mecánica. Solamente resta extraer las soluciones individuales del árbol purgado y ordenado para que el método ejecutor pueda ponerlas en práctica ordenadamente.

Si bien este trabajo no se ocupa de describir el funcionamiento del método ejecutor, es preciso comentar que en el caso de que existan múltiples soluciones, no se ejecutan todas si no es absolutamente necesario. Por el contrario, en el preciso momento en que una de las soluciones genera sin problemas el producto buscado, el proceso se da por finalizado (nótese que los métodos incluidos en una solución pueden fallar, por ejemplo, por problemas de convergencia en un cálculo iterativo; por esta razón, es necesario suponer que las soluciones obtenidas por el diseñador son susceptibles de finalizar incorrectamente).

6. Breve discusión sobre la caracterización del comportamiento

Para poder caracterizar los métodos incluidos en el "framework" clásico de manera que la clase inteligente sepa cual es su función e integrarlos en soluciones a problemas más complejos, es preciso formalizar la manera en que se describe la semántica de aquellos. En esta Sección se hace un breve resumen del método utilizado; para más detalles, véase [4].

En este trabajo se ha huido de las técnicas clásicas de especificación formal basadas en la lógica por ser estas no solo insuficientes para capturar esa semántica –que ya es razón suficiente- sino también por la complejidad que su utilización comporta.

Es por ello que se ha atacado el problema en conjunción con las armas que provee la orientación a objetos. En particular, se sigue, aunque no la implanta completamente, la aproximación descrita en [6], la notación (y el concepto) BON ("Business Object Notation" o Notación de Objetos de Negocio) así como el paradigma de "Diseño por contrato" (véase [2,3]).

El concepto detrás de BON usa como piedra angular los métodos de *consulta* de los objetos de una determinada clase para especificar las pre- y postcondiciones de sus métodos. Los métodos de consulta son aquellos que dan información sobre el estado de un objeto, en contraposición a los que lo alteran. Por ejemplo, la clase "Imagen" podría tener toda una panoplia de consultas disponibles para obtener información sobre las propiedades de los objetos de este tipo: "es RGB?", "Número de bandas?", "Tamaño en filas de la imagen?", "Está geocorregida?", etc. BON utiliza directamente esas consultas cuando especifica, por ejemplo, una precondición como podría serlo la siguiente:

Imagen.EsRGB and Imagen.TamañoX > 0 and Imagen.TamañoY>0

Esta precondición establece que el objeto "Imagen" ha de estar en formato RGB y que su tamaño en X e Y ha de ser mayor que cero en ambos casos.

La "belleza" de BON es la simplicidad con la cual se consigue especificar una semántica sumamente compleja. Las condiciones impuestas sobre Imagen en el ejemplo anterior serían prácticamente imposibles de especificar utilizando exclusivamente lógica formal –y si fuera posible, la especificación resultante sería extremadamente compleja! Además, este tipo de especificación no adolece de los problemas clásicos de otras notaciones. El más importante es la no existencia de vacíos entre especificación y programación. Es decir, todo lo que se especifica tiene una correspondencia directa con lo que se programa y viceversa.

Esto es así porque los operandos utilizados se corresponden *directamente* con métodos de consulta de la clase. Yendo aún más lejos: si por alguna razón aparece la necesidad de una nueva propiedad para ser utilizada en la especificación, lo único que se necesita es programar el correspondiente método de consulta. Esto hace que la notación sea totalmente escalable, y que nunca existan situaciones en las que sea imposible especificar el comportamiento.

Dentro del contexto de la "framework" inteligente se sigue esta aproximación para caracterizar los métodos existentes. En concreto, para cada método se incluye la lista de propiedades de todos y cada uno de sus objetos de entrada y de salida.

Para los objetos de entrada estas propiedades están estableciendo las *precondiciones* del método, es decir, qué características han de satisfacer esos objetos para que el método pueda ser utilizado correctamente. En el caso de las salidas, se está informando de cuales serán las propiedades que satisfarán los objetos creados. De esta manera queda caracterizada perfectamente la semántica de los solvers.

Por ejemplo, imagínese que se dispone de un método que a partir de una imagen digitalizada y un modelo digital de terreno es capaz de producir una ortofoto, es decir, una imagen geocorregida. Las condiciones que ha de cumplir la imagen de entrada podrían ser (1) que fuera en color (RGB) y en un formato de un determinado fabricante, que denominaremos "Formato X". Por otra parte, el modelo digital del terreno ha de estar organizado en malla. Si esto se satisface y se invoca el método en cuestión,

garantizará que la imagen de salida estará geocorregida y que, por ejemplo, el formato de salida será otro diferente, que denominaremos "Formato Y". Utilizando la notación BON, las precondiciones y postcondiciones serían las siguientes:

Precondiciones:

Objeto imagen: Imagen1.EsRGB and Imagen1.EsFormatoX
Objeto modelo digital del terreno: ModeloDigital1.OrganizadoEnMalla

Postcondiciones

Objeto imagen: Imagen2.EsRGB and Imagen2.EsFormatoY and Imagen2.EstaGeocorregida

Evidentemente, la clase Imagen ha de disponer de los métodos EsRGB, EsFormatoX, EsFormatoY así como EstaGeocorregida. De esta manera, el método, al ejecutarse, puede, efectivamente, comprobar que sus objetos de entrada cumplen con los prerrequisitos necesarios. En el caso de la clase ModeloDigital, ha de existir el método OrganizadoEnMalla.

Este tipo de especificación es el mismo que se utiliza para describir el objetivo de las factorías a diseñar.

Volviendo al tema del funcionamiento del método diseñador de la clase inteligente, ha de quedar ahora claro como funciona su primera etapa, cuando, utilizando exclusivamente las descripciones formales, localiza métodos capaces de crear un determinado tipo de objetos. En particular, si se quisiera rectificar una imagen, buscaría todos aquellos métodos que generasen un objeto con la propiedad EstaGeocorregida.

7. Conclusión

El método diseñador descrito hasta ahora es una alternativa "inteligente" a herramientas como Visual Factory Builder incluídas en la Visual Factory Suite. Por medio de un proceso de normalización y caracterización de los métodos disponibles así como de la formalización del conocimiento, es posible llegar a diseñar factorías automáticamente, las cuales recogen toda la experiencia de los expertos de la disciplina y, por tanto, expresan el saber hacer acumulado durante, posiblemente, años de trabajo.

Evidentemente, la formalización del conocimiento así como la caracterización de los métodos disponibles en el "framework" es un esfuerzo más que considerable. Pero aparte de las ventajas que esto pueda originar por su utilización en entornos como el aquí descrito (VFS, "frameworks" inteligente) un esfuerzo orientado a, si más no, clarificar cuales son las formas de proceder en un determinado ámbito negocio no puede ser más que positivo. ¡Al menos, éste es el espíritu de la certificación de calidad ISO 9000!

Existe un prototipo de este método diseñador que prueba la validez del concepto y que se describe ampliamente en [4]. El siguiente paso será conseguir su integración con herramientas industriales destinadas a la explotación de sistemas de producción masiva por lotes y, en particular, complementar el editor visual de factorias, Visual Factory Builder de la Visual Factory Suite.

Bibliografía

[1] Colomina, Ismael; Navarro, José. El sistema de generación de ortofotos digitales del ICC. En *Sistemas cartográficos digitales Españoles, productos y desarrollos. Primera jornada técnica de la SECFT*. Madrid, 1989.

[2] Meyer, Bertrand. Applying "Design by Contract". *IEEE Computer* 25(10):41-55. Octubre 1992.

[3] Meyer, Bertrand, *Object Oriented software construction*. Prentice Hall, 2ª edición. 1997.

[4] Navarro, José. Object-oriented technologies and beyond for software generation and integration in Geomatics. *Monografies techniques de l'Institut Cartogràfic de Catalunya*, num. 6. 1998. Barcelona, España.

[5] Navarro, José. The Visual Factory Suite: Facing evolving mass production in spatial data processing environments. *XIXth Congress of the ISPRS*. 2000, Amsterdam, Holanda.

[6] Waldén, Kim; Nerson, Jean-Marc. *Seamless object-oriented software architecture. Analysis and design of reliable systems*. The Object Oriented Series, Prentice Hall. 1995