UN FRAMEWORK PARA GENERACIÓN DE TUTORIALES

Zulema B. Rosanigo[1]; Alicia Paur[2]; Pedro Bramati[3]; Alfredo Ortega[4]; José P.Cerra[5]

Facultad de Ingeniería – Sede Trelew – U.N.P.S.J.B. Te-Fax (02965) 42 84 02

RESUMEN

El presente trabajo es parte del proyecto de investigación en el área de informática educativa denominado "Construcción de tutoriales basados en componentes reusables" que tiene por objeto facilitar el desarrollo de material didáctico en general, y de tutoriales en particular, con capacidad de ser utilizados a través de Internet y basados en componentes reusables que soporten un conjunto de comportamientos estándares, independientemente de la funcionalidad específica para la que han sido diseñados.

Se trata de diseñar una herramienta, que a partir de algunos componentes –desarrollados especialmente para el dominio de aplicación– el docente pueda construir tutoriales del tipo enseñanza paso a paso, para que el alumno pueda ejercitarse tanto como lo necesite.

Este tutorial que acaba de crear, se convierte en un nuevo componente del dominio y por lo tanto puede ser utilizado para la creación de otro, debiéndose adaptar en forma inteligente al nuevo contexto.

En este artículo mostramos la arquitectura y cuestiones de diseño de algunos componentes.

Palabras claves:

Tutorial – componente reusable – framework – patrón de diseño

[1] Ingeniera Civil – Analista Programador Universitario – Magister en Ingeniería de Software - Investigador Cat. III - Profesor Asociado D.E. brosanigo@infovia.com.ar

[2] Analista Programador Universitario - Investigador Cat. V – J.T.P. D.S.E. - apaur@topmail.com.ar

[3] Ingeniero Civil – Investigador Cat. IV - Profesor Titular D. S.E.. bramati@infovia.com.ar

[4] Analista Programador Universitario - Alumno de la Licenciatura en Informática cachito@ortega.net.ar

[5] Alumno de Analista Programador Universitario– cerraguza@infovia.com.ar

1- INTRODUCCIÓN

La realidad universitaria actual presenta una situación dicotómica: por un lado se realizan recortes en la cantidad de horas de la currícula y por el otro, se necesitarían más para suplir las deficiencias con las que llega el alumnado. Una de las inquietudes de todos los equipos docentes consiste en encontrar, investigar y probar métodos que sean más adecuados para transmitir los conocimientos de un modo eficiente, intentando adaptar los contenidos de las materias a las necesidades presentes y futuras de los alumnos.

La posibilidad de individualizar el proceso de enseñanza-aprendizaje que nos ofrece la computación representa importantes ventajas, pues puede permitirnos optimizar las capacidades de cada individuo al no tener que imponerles a todos los estudiantes el mismo ritmo, lo que es, para los más hábiles, motivo de aburrimiento, y para los que tienen mayor dificultad, frustración y decepción en dicho proceso.

La utilización de tutoriales basados en multimedia es una opción a tener en cuenta en todas las materias de la currícula de cualquier carrera universitaria. Su incorporación al proceso de enseñanza-aprendizaje exige ingenio y creatividad, y sobre todo, contemplarlo "de otro modo" ya que afectará a todos los elementos que componen a ese proceso.

A partir de la utilización de tutoriales, se espera resolver una de las dificultades que se plantea en el aprendizaje tradicional de algunas materias demasiado técnicas, donde en forma usual el alumno se encuentra con la resolución impresa en un solo paso, con todos los procesos intermedios obviados. Al poder seguir el proceso “paso a paso”, se eliminará la dificultad antes expuesta.

Nuestro objetivo es el desarrollo de una arquitectura de software reusable para facilitarle al docente la construcción de nuevos tutoriales, los cuales, además, puedan ser reutilizados como nuevos componentes para la construcción de otros. Con su uso, el alumno podrá ejercitarse tanto como lo necesite, hasta adquirir los conocimientos y destreza necesarios.

La idea es diseñar un framework para la generación de tutoriales: aportar la estructura y funcionalidad de los componentes comunes, establecer las interacciones entre los objetos intervinientes y definir las abstracciones fundamentales y sus interfaces. En el framework quedan especificados los aspectos de los tutoriales que se mantienen estables en todas las aplicaciones y se proveen mecanismos para poder expresar las variaciones que sean necesarias.

En las secciones siguientes, se describe cómo funciona un tutorial para enseñanza “paso a paso”, la arquitectura de la herramienta propuesta y sus abstracciones fundamentales.

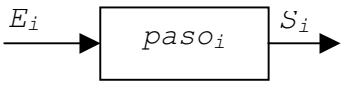
2- CÓMO FUNCIONA UN TUTORIAL PARA ENSEÑANZA “PASO A PASO”?

En un tutorial de este tipo se pretende enseñar mediante un ejemplo, a resolver un problema, mostrando a partir de un conjunto de datos iniciales, la forma de llegar al resultado mostrando todos los pasos intermedios necesarios.

Si E_0 es el conjunto de datos iniciales y S es el resultado esperado, podemos afirmar que $S = f(E_0)$ donde f representa la función o algoritmo que resuelve el problema y cuyos pasos detallados se quieren enseñar.

Lo que se pretende mostrar son las transformaciones necesarias a realizar sobre E_0 para producir la salida S. Cada una de estas transformaciones es un paso del tutorial y habrá que realizarlas en un cierto orden.

Cada paso también tiene sus datos de partida y produce un resultado. Puede verse como un esquema de caja negra, donde las entradas del paso *i* se encuentran entre las salidas de los pasos anteriores y/o los datos iniciales, en forma directa o mediante operaciones sobre ellos.

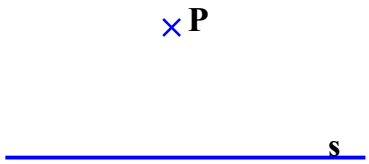


$$E_i \subset E_0 \cup \{S_1, S_2, \ldots, S_{i-1}\}$$
$$S_i = f_i(E_i)$$

2.1- Ejemplo de tutorial paso a paso:

Problema: Trazar la perpendicular a una recta **s** y que pase por el punto **P**

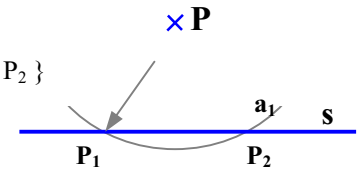
Entrada: $\mathbf{E_0}$ = { s, P}



Pasos:

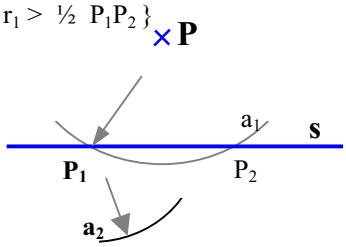
1 Con centro en **P** trazar un arco $\mathbf{a_1}$ que corte a la recta **s** en dos puntos: $\mathbf{P_1}$ y $\mathbf{P_2}$

E_1 = { P, s}
S_1 = { a_1 , P_1 , P_2 }



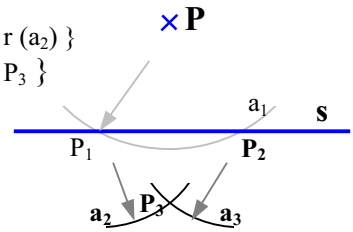
2 Con centro en $\mathbf{P_1}$ trazar un arco $\mathbf{a_2}$ con radio $r_1 > \frac{1}{2} P_1P_2$

E_2 = { P_1, $r_1 > \frac{1}{2} P_1P_2$ }
S_2 = { a_2 }



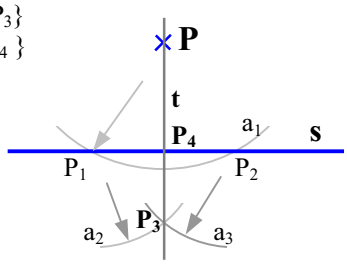
3 Con centro en $\mathbf{P_2}$ trazar un arco $\mathbf{a_3}$ con el mismo radio que $\mathbf{a_2}$ y que lo corte en $\mathbf{P_3}$

E_3 = { P_2, r (a_2) }
S_3 = { a_3 , P_3 }

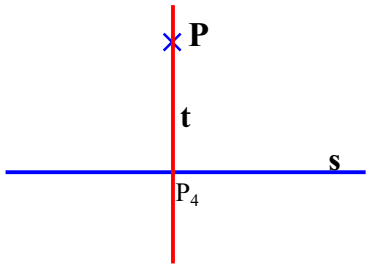


4 Trazar la recta **t** que pasa por **P** y $\mathbf{P_3}$ cortando a **s** en $\mathbf{P_4}$

E_4 = { P, P_3}
S_4 = { t , P_4 }

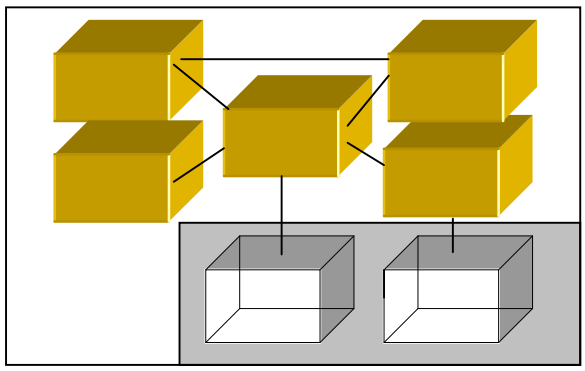


Resultado: **S** = { t }



3- GENERADOR DE TUTORIALES

El framework forma el esqueleto de la aplicación, provee la funcionalidad común, define las abstracciones fundamentales y sus interfaces, establece las interacciones entre los objetos, dejando en determinados lugares espacios en blanco o puntos de articulación (hot spot). Cada espacio se refiere a los aspectos de los tutoriales que pueden variar de una aplicación a otra, y es allí donde el framework debe proveer flexibilidad.



Para realizar una aplicación de generación de tutoriales en un dominio en particular, basta con incluir los componentes que satisfagan las situaciones requeridas por la aplicación. Por ejemplo, si pretendemos crear un generador de tutoriales en el dominio de la Geometría deberíamos incluir componentes que representen los entes geométricos, sus gestores y las operaciones primitivas que pueden realizarse sobre ellos.

La aplicación que permita generar tutoriales debe ser interactiva, con una interfaz gráfica para visualizar cada paso del tutorial y permitir tratar los eventos generados por las acciones del usuario. Contará con un editor de tutoriales bastante semejante a un editor gráfico, un panel de dibujo o pizarra, barra de herramientas que permitan indicar el tipo y valor de los datos iniciales (Punto, Recta, Arco,...), pasos intermedios (arco que corta a recta, arco con centro y radio, arco con centro y radio que corta a arco,...), selección, desplazamiento, etc. La barra de herramientas variará según el dominio, y aún cuando el dominio de aplicación del tutorial no sea gráfico, siempre necesitará de la pizarra para reflejar los datos iniciales y las transformaciones necesarias para llegar al resultado del problema. En la pizarra se "dibuja" cada paso del tutorial.

El modelo arquitectónico en que se basa la aplicación es el Model-View-Controller [1], que divide el problema en tres componentes: el modelo que contiene el corazón de la funcionalidad, la vista que despliega la información al usuario y el controlador que maneja la entrada del usuario. La vista y el controlador definen la interfaz gráfica.

3.1- Diagrama general de clases

El siguiente diagrama muestras las clases más importantes y sus relaciones:

TutorialEditor define la interface para coordinar los diferentes objetos que participan en un editor. Desacopla los participantes del editor y actúa como un coordinador de ellos (patrón mediator [4]). Para independizar el editor de sus vistas, se aplica el patrón Observer [4].

TutorialView muestra la Pizarra y escucha sus cambios. Recibe los eventos de la interfaz de usuario y las delega a la herramienta actual. La respuesta depende de la herramienta activa en ese momento.

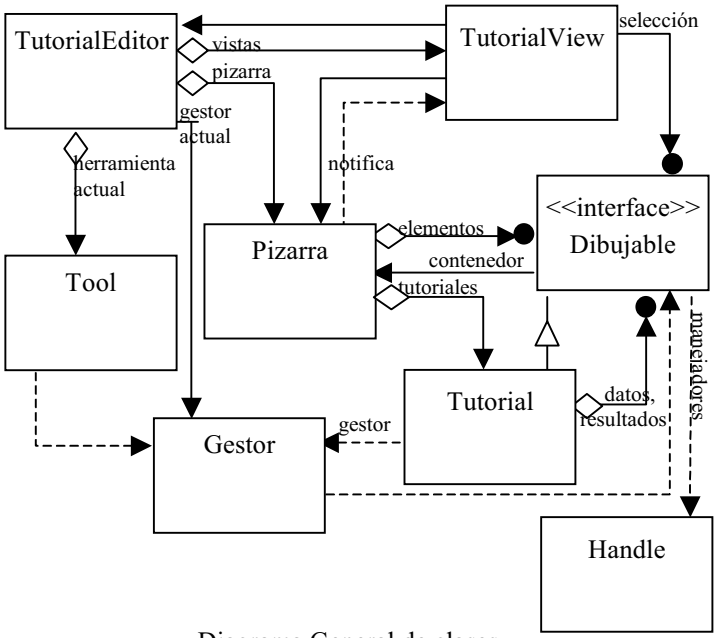


Diagrama General de clases

La reacción del editor ante los eventos depende del estado en el que se encuentre el editor. Para solucionar este problema se utiliza el patrón State [4]. El patrón State permite a un objeto cambiar su comportamiento cuando cambian sus atributos internos.

Las herramientas son creadas una vez y reutilizadas. Son inicializadas y desinicializadas con activate() y deactivate() respectivamente.

La Pizarra es un contenedor para Tutoriales, envía los eventos de sus cambios a los oyentes cuando una parte de su área fue modificada.

Tutorial es una de las abstracciones fundamentales. La clase abstracta Tutorial provee el comportamiento genérico y la interface que deben cumplir sus subclases. Conoce cómo representarse gráficamente. Para gestionar los datos iniciales (entradas) proporciona Gestores.

Un Dibujable tiene un conjunto de manejadores que manipulan su forma o sus atributos. Es la interfaz que deben cumplir los objetos del dominio que pueden actuar de datos iniciales o resultados de un tutorial para ser representados gráficamente.

Los Handles son utilizados para cambiar un dibujable por manipulación directa. Los manejadores conocen sus propias figuras y proporcionan métodos para localizar y ubicar el manejo sobre la figura y seguir la pista a los cambios.

3.2- Algunos requerimientos funcionales

- Contemplar dos modalidades de uso bien definidas: creación de un tutorial y uso de un tutorial.
- El trabajo de creación de un tutorial mediante la herramienta, debe resultarle al docente "tan natural" como si lo desarrollara manualmente, y en general, mucho más sencillo. Esto implica que muchas tareas pueden y deben estar automatizadas.
- El tutorial que acaba de crear debe poder convertirse en un nuevo componente del dominio y por lo tanto, podrá ser utilizado para la creación de otro, debiéndose adaptar en forma inteligente, al nuevo contexto. Esto implica que debe existir un mecanismo especial de persistencia para los tutoriales creados.
- En modalidad de ejecución de un tutorial, puede ser practicado individualmente tantas veces como se quiera, en forma completa o parcial, permitiendo volver atrás cuando el alumno lo requiera.

4- ABSTRACCIONES FUNDAMENTALES

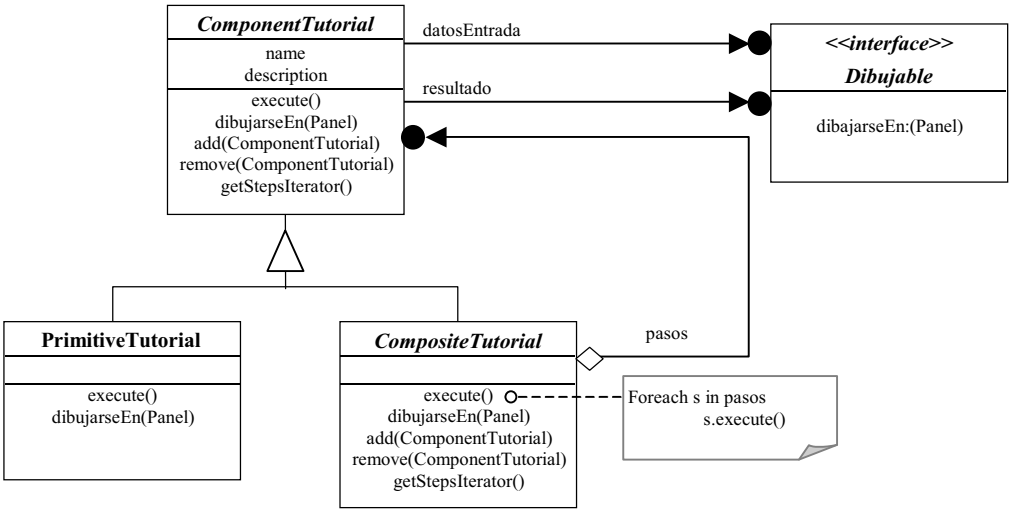
Muchas abstracciones son similares a las que se encuentran en un editor gráfico, vamos a concentrarnos en aquellas que son propias de este generador de tutoriales.

4.1- Tutorial

El tutorial consta de entrada, pasos y salida. La entrada y la salida son contenedores de objetos propios del dominio de aplicación. Cada paso del tutorial puede ser una primitiva (simple), u otro tutorial (compuesto). Por ejemplo, para resolver el problema de trazar la paralela a una recta s y que pase por el punto P, uno de los pasos, consiste en trazar la recta perpendicular a la recta s pasando por el punto P. Cualquier tutorial creado por el Generador de Tutoriales es un tutorial compuesto, ya sea por pasos simples o por pasos que también son compuestos.

Patrón de diseño: Composite

El composite nos permite tratar uniformemente los objetos simples, de los compuestos. El cliente utiliza la interfaz del *Component* para actuar con los objetos de la estructura del patrón sin necesidad de realizar una diferencia entre objetos simples y compuestos. Si el objeto es simple, el requerimiento es tratado directamente, y si es compuesto, el *Composite* delega el requerimiento a sus componentes hijos.



Tutorial

¿De qué tipo son las entradas y las salidas?

Desde el punto de vista de los datos del problema, el tipo de entrada y de salida es propia del dominio de aplicación. Desde el punto de vista del Generador de tutoriales, ambas comparten el saber dibujarse sobre el panel de la pizarra. Por lo tanto, podemos abstraer este comportamiento en una interface *Dibujable*, y hacer que los objetos del dominio que pueden ser entradas o salidas implementen la Interface Dibujable. De esta manera el diseño y la implementación quedan separadas y la abstracción es independiente de cualquier detalle de implementación.

Cada tutorial es responsable de gestionar sus propios datos de entrada.

4.2- ¿Cómo modelar la gestión de datos?

Es importante analizar si el proceso de gestión de datos es el mismo para datos iniciales, E_0, de un tutorial, que para las entradas Ei de los pasos de ese tutorial. Nos planteamos y contestamos las siguiente preguntas:

a) Para la *generación de un nuevo tutorial*, los datos iniciales y las entradas de un paso se obtienen de la misma forma?

   Modo creación: Es el único modo en el que se da esta situación. La entrada de datos iniciales y de cada paso, se realiza seleccionando de la barra de herramientas el botón adecuado que permita crearlo o seleccionarlo si ya existe. Requiere interacción del usuario (docente).

b) Para una *instancia de tutorial*, los datos iniciales y las entradas de un paso se obtienen de la misma forma?

Modo creación: Esta situación sólo puede darse si la instancia es paso de un tutorial que se está creando. En este caso, la entrada E_0 de esta instancia debe obtenerse atendiendo interacciones del usuario. Cada paso de la instancia, debe saber adaptar o calcular sus entradas adecuadamente, sin interacción del usuario.

Modo uso: Si es un paso de otro tutorial, no hace falta indicar cuál es la entrada: tanto el paso como los subpasos recalculan sus entradas y no hay interacción del usuario.

Si se ejecuta solo, se gestionan los datos de entrada y se recalculan los datos de cada paso. Requiere interacción del usuario (docente o alumno).

Se pueden observar entonces, dos casos diferentes:

- Los datos se obtienen atendiendo eventos de usuario.
- Los datos se recalculan en función de datos existentes.

El Generador de tutoriales y cualquier otro cliente, sabe antes de crear una instancia de un tutorial, si éste debe o no gestionar los datos de entrada: Si el cliente es el Generador de tutoriales sabe que tanto en modo creación (la instancia es un paso del que se está creando), como en modo uso (la instancia es la elegida para ser ejecutada), requiere de la interacción del usuario para gestionar sus datos iniciales. Si el cliente es otro tutorial, éste necesita ir creando las instancias correspondientes a sus propios pasos (que también son tutoriales), para los cuales debe recalcular sus entradas en función de los pasos anteriores.

Por lo tanto, el constructor del tutorial podría estar sobrecargado para soportar ambos casos: constructor nulo (requiere interacción del usuario para gestionar sus datos iniciales) y constructor con parámetros (correspondientes a las entradas iniciales conocidas). En el primer caso, se delega la responsabilidad de gestión en una jerarquía de clases separadas aplicando el patrón *Factory Method* para conectarlo con el tutorial, quien decide que tipo de objeto crear.

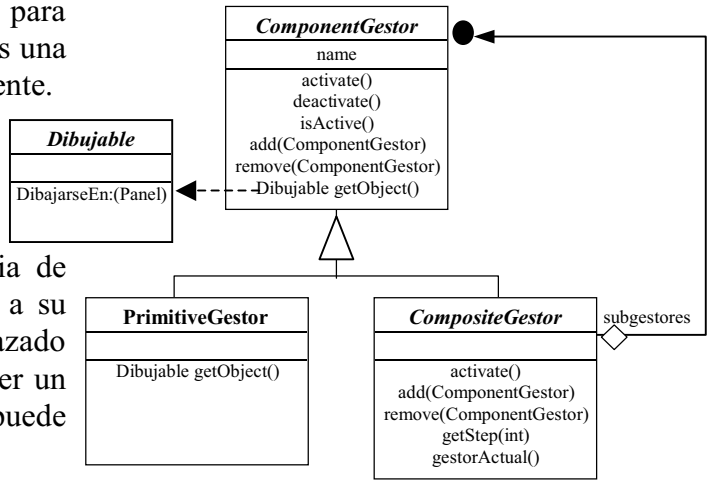
Primero vamos a describir la jerarquía de clases que maneja la gestión. Luego, veremos cómo quedan vinculadas al aplicar el Factory Method.

4.2.1- Gestor

Un gestor es un objeto que diligencia o tramita los datos necesarios para crear el objeto requerido u obtenerlo mediante selección si ya existe. La gestión puede hacerla atendiendo eventos de usuario o porque recibe los parámetros necesarios para inicializarlo. No siempre el objeto obtenido es una nueva instancia, puede ser una instancia existente.

Para cada tipo de objeto de entrada del dominio, hay un gestor asociado.

Un gestor puede ser simple o compuesto. En este último caso formado por una secuencia de gestores, cada uno de los cuales puede ser a su vez, simple o compuesto. En el tutorial del trazado de la perpendicular, el Gestor necesita obtener un punto y una recta, y la gestión de una recta puede hacerse mediante la gestión de dos puntos.



Gestor

Patrón de diseño: Composite

Con la aplicación de este patrón los gestores simples y compuestos son tratados de modo uniforme.

4.2.2- ¿Cómo se relaciona el Gestor y el Tutorial?

Los datos de entrada de un tutorial son gestionados por el Gestor. Cuando se selecciona un tutorial para su ejecución, éste debe delegar la gestión de sus datos iniciales al Gestor.

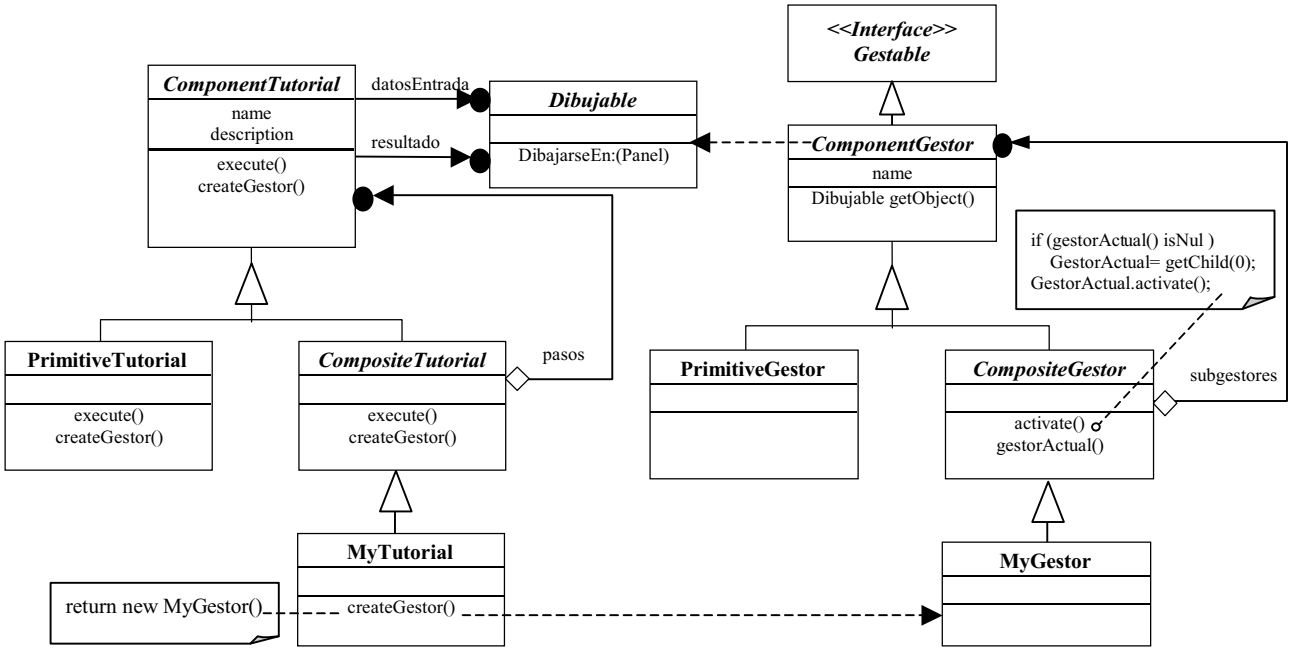
¿El Gestor es un componente del tutorial? ¿O cada tutorial crea al gestor cuando lo necesita?

No es necesario que el gestor esté presente durante toda la vida de la instancia de tutorial, una vez finalizada la obtención de los datos, ya no se necesita. Tampoco es necesario si se conocen los datos iniciales del tutorial. Por lo tanto, cada tutorial creará al gestor cuando le sea necesario o se lo soliciten.

Sólo las subclases concretas de ComponentTutorial poseen elconocimiento de qué tipo gestor crear.

Patrón de diseño: *Factory method*

Define una interface para crear un objeto, dejando a las subclases decidir el tipo específico. Permite delegar la responsabilidad de instanciación a las subclases.



Tutorial y Gestor

La clase tutorial tendrá un factory method, createGestor(), que devuelve una instancia de Gestor. Cada tutorial sabe qué tipo de gestor crear. En general, se trata de un gestor compuesto, ya que los tutoriales suelen requerir varios datos iniciales, y por cada dato de entrada, debe prever un gestor.

Cuando es el TutorialEditor quien crea una instancia de tutorial lo hace con el constructor nulo y a esa instancia le pide su gestor y lo activa. El gestor se mantiene activo hasta que finaliza su gestión de datos. Si hay un gestor activo, las interacciones de usuario son retransmitidas al gestor. Los gestores compuestos, van activando a sus subgestores cada vez que un subgestor finaliza su gestión.

La interfaz Gestable establece el protocolo de retransmisión de eventos. Los gestores deben implementar Gestable. La clase abstracta ComponenGestor provee una implementación por defecto.

4.3- Cómo manejar la persistencia de los tutoriales creados por el usuario?

La persistencia provista por las clases no alcanza para resolver este problema, ya que el objetivo de la herramienta Generadora de tutoriales en un dominio en particular, es que cada tutorial que se acaba de crear, se convierta automáticamente en un nuevo componente del dominio sin que haya que compilar. El docente crea un nuevo tutorial y lo guarda. Se pretende que ese tutorial guardado pueda ser usado en forma transparente para el docente, sin distinción de que se trate de un tutorial ya provisto por la herramienta (existe un ".class") o uno almacenado en un archivo.

Nos enfrentamos a la situación en que no sólo el alumno debe aprender sino que el sistema que estamos diseñando también debe "aprender" a utilizar los tutoriales creados, de la misma manera que sabe utilizar las primitivas.

Primera aproximación: *Base de datos de órdenes y propiedades y un intérprete*.

Problema: Para que sea genérico, el lenguaje de órdenes y su intérprete resulta bastante complejo de definir. De otra manera, queda dependiente del dominio de aplicación en que se aplicarían los tutoriales y para cada dominio habría que definirlo.

Segunda aproximación: *Objetos Serializables*.

Los objetos generados durante el proceso de creación del tutorial (entradas y salidas de cada paso) se guardan en la misma secuencia de generación. En modo uso, a medida que se los recupera se les envía el mensaje dibujarse. La principal ventaja de este mecanismo es que es simple, genérico y que no requiere esfuerzo de programación.

Problema: Permite la ejecución del tutorial tantas veces como se necesite pero no se adapta a nuevos contextos, no pudiendo ser utilizado como paso de otro tutorial. Sirve para que el alumno aprenda a resolver el problema en el tiempo que le sea necesario, pero para resolverlo con otros datos, debería generarse un nuevo tutorial.

Mejorando esta última aproximación: *Pasos Inteligentes Serializables*

Necesitamos que:

- *Cada tutorial tenga un mecanismo de persistencia.*
- *Cada tutorial sepa autoejecutar los pasos en función de las entradas del nuevo contexto en que se ejecuta y las salidas de los pasos anteriores.*

Necesitamos diseñar una forma de guardar el tutorial en un medio externo y recuperarlo con la inteligencia necesaria que le permita poder adaptarse al uso que se le quiera dar, con o sin gestión de sus propias entradas, y siempre recalculando las entradas de sus pasos componentes.

En modo creación, cada elemento de la barra de herramientas o del menú de elecciónde pasos que dispone el docente para crear un tutorial, está asociado a un gestor, quien se encarga de atender los eventos de usuario para poder producir el objeto pedido. En modo uso, ese tutorial creado por el docente, debe "recordar" cómo se obtienen las entradas de cada uno de sus pasos.

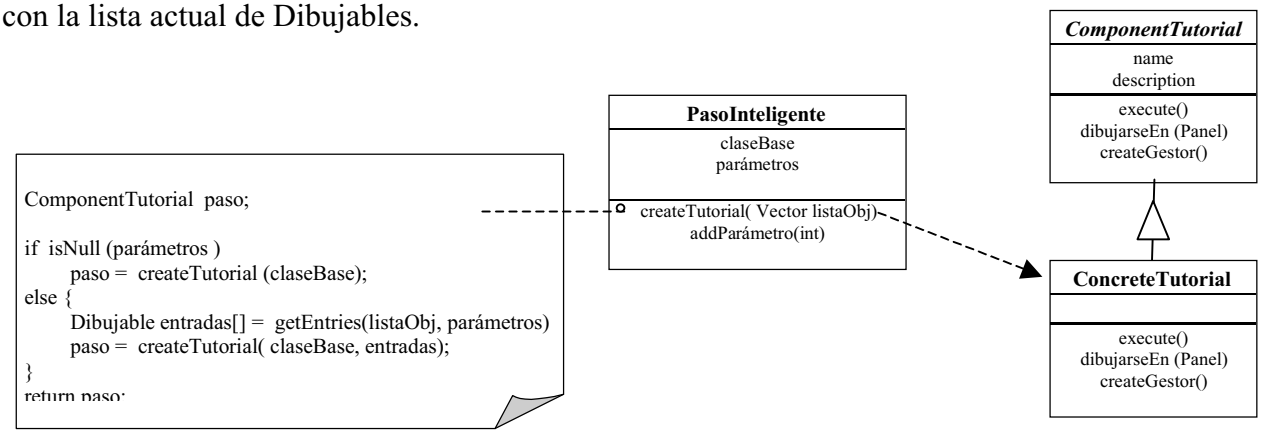
4.3.1 Paso Inteligente

Un paso inteligente es un objeto serializable que guarda referencias acerca de la clase correspondiente a un paso del tutorial que se está creando, y del orden relativo en que se encuentran creadas sus entradas respecto del tutorial que forma parte. Tiene un factory method (createTutorial) con el que se puede crear la instancia del tutorial que representa con los datos que éste necesita.

En modo creación y en forma paralela a las acciones del docente para indicar los pasos de un tutorial, el paso inteligente tiene en cuenta si en el proceso de gestión de datos, se crea un nuevo objeto o se utiliza un objeto existente, y registra esta información y el orden relativo en que ese objeto de entrada se encuentra dentro de la lista de objetos existentes al momento (entradas iniciales o salidas de pasos anteriores).

Cuando el docente finaliza la tarea de crear un nuevo tutorial y da la orden de guardarlo, el Editor de tutoriales, guarda la secuencia de pasos inteligentes con el mecanismo previsto para objetos serializables.

Cuando se desea utilizar este tutorial, el Editor de tutoriales crea una instancia de Tutorial Compuesto al que le va adicionando los tutoriales (pasos) devueltos por cada Paso Inteligente recuperado. Para ello, a medida que recupera un PasoInteligente, le envía el mensaje createTutorial con la lista actual de Dibujables.



Paso Inteligente

Según la información que ha guardado acerca de los objetos utilizados en el momento de su creación, sabrá si debe esperar interacción del usuario o no. Y en este último caso, también sabe en qué orden puede encontrar, dentro del vector de objetos dibujables existentes hasta el momento, aquellos que necesita para inicializar la entrada E_0 del tutorial que representa.

5- CONCLUSIONES

En este trabajo se han mostrado algunas de las abstracciones que se están utilizando en el diseño de framework para generación de tutoriales.

Los patrones de diseño y Java son una combinación muy potente para la construcción de este tipo de herramienta.

La utilización de los patrones de diseño proporciona una arquitectura flexible, independiente y extendible y ayudan en la documentación (pattlets).

El patrón Gestor coloca el comportamiento asociado a la gestión de datos iniciales de un tutorial dentro de un objeto, logrando desacoplar el problema de la clase tutorial. Las jerarquías de gestores y tutoriales pueden evolucionar en forma independiente.

Paso Inteligente encapsula el conocimiento y provee los mecanismos que permiten lograr que los tutoriales creados por el usuario tengan una forma de persistencia tal, que puedan adaptarse al ser utilizados en nuevos contextos.

6- BIBLIOGRAFÍA

[1] Buschmann F., Meunier R., Rohnert H., Sommerland, P., Stal, M. *Pattern-Oriented Spftware Architecture: a system of patterns.* Ed. Wiley 1996

[2] Cooper, James W. - *Java Design Patterns: A Tutorial*, 1998 – Addison Wesley

[3] Eckel, Bruce. *Thinking in Java* 2nd ed. ISBN 0-13-027363-5- Prentice Hall

[4] Gamma, Eric; Helm, Richard; Johnson, Ralph and Vlissides, John, *Design Patterns. Elements of Reusable Software*, Addison-Wesley, 1995

[5] Johnson, R. *Documenting frameworks using patterns* – OOSPLA'92

[6] Rosanigo, Z.B., Bramati, P., Paur, A., *Construcción de tutoriales basados en componentes reusables*. Actas Workshop de Investigadores en Ciencias de la Computación 2002 – Bahía Blanca.

[7] Rosanigo, Z.B., Paur, A., Bramati, P. *Metodología de desarrollo de software educativo*. Actas de VI Congreso Internacional de Ingeniería Informática ICIEY2K Fac. de Ingeniería, U.B.A.- Buenos Aires – 2000