Implementación de componentes multi-propósito y un entorno de desarrollo

14 de julio de 2003

Artículo enviado al IX Congreso Argentino de Ciencias de la Computación

Autores:

LIC. ALFREDO ORTEGA [1]
JOSÉ PABLO CERRA [2]
MG. BEATRIZ ROSANIGO [3]
ING. PEDRO BRAMATI [4]
A.P.U. ALICIA PAUR [5]

Facultad de Ingeniería – Sede Trelew - Universidad Nacional de la Patagonia San Juan Bosco

Resumen:

Durante el proyecto de investigación No 383 surgió la necesidad de un sistema de componentes reusables multi-propósito, junto con un entorno integrado de desarrollo que los utilice. Existiendo muchas tecnologías disponibles se optó por extender el modelo Java-Beans de Sun.
Este artículo describe las estructuras y métodos utilizados para lograr una solución flexible tanto para el usuario del sistema como para el programador de componentes.

Palabras claves:

Componentes reusables - Java-Beans – Framework - Tutorial

[1] Licenciado en Informática cachito@ortega.net.ar
[2] Alumno de Analista Programador Universitario – cerraguza@yahoo.com.ar
[3] Ingeniera Civil – Analista Programador Universitario – Magister en Ingeniería de Software - Investigador Cat. III - Profesor Asociado D.E. brosanigo@infovia.com.ar
[4] Ingeniero Civil – Investigador Cat. IV - Profesor Titular D. S.E.. bramati@infovia.com.ar
[5] Analista Programador Universitario - Investigador Cat. V – Profesora Adjunta D.S.E. - apaur@topmail.com.ar

1. Introducción

El proyecto de investigación Nro. 383 [PI+2001] tiene por objetivo facilitar la generación de material didáctico en general, y de tutoriales del tipo "enseñanza paso a paso"en particular,. Para ello, se propuso diseñar un framework basado en componentes que permita construir y ejecutar tutoriales en cualquier dominio del conocimiento, y de forma tal, que esa tarea resulte sencilla y simple para el docente o usuario que lo utilice.

Con un tutorial 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 individualizando 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.

El generador de tutoriales está pensado para su utilización por usuarios sin conocimiento de programación, por lo que se necesita asistencia adicional por parte del software para efectuar la interrelación y ensamblaje de los componentes en un tutorial interactivo, tarea que puede llegar a ser muy compleja. No es aceptable utilizar la aproximación que siguen otros entornos orientados a componentes, tales como *Visual Basic*, ya que en este caso se trata de sistemas para usuarios avanzados con conocimiento en programación, en los que la inter-relación entre los componentes se realiza programáticamente.

Para afrontar dicha complejidad se decidió diseñar e implementar un IDE (Entorno Integrado de Desarrollo) visual, al que hemos denominado TutGen, junto a un conjunto de componentes:

- ✓ **TutorialEditor**, un editor de tutoriales que coordina los diferentes objetos que participan en un editor. Desacopla los participantes del editor y actúa como un coordinador de ellos.
- ✓ **TutorialView** que gerencia toda la visualización, 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.
- ✓ **Pizarra** que es es el panel donde se despliega el tutorial.
- ✓ **Tool**, las herramientas para edición y ejecución de tutoriales.
- ✓ **Tutorial**, una de las abstracciones fundamentales, provee el comportamiento genérico de todo tutorial, puede ejecutarse paso a paso, retroceder o avanzar un paso, sabe representarse, gestionar sus datos, calcular el resultado.

Para crear tutoriales en un dominio en particular, se requiere definir los elementos que pueden ser datos y las acciones primitivas que pueden efectuar transformaciones de esos datos, generando nuevos elementos. Este comportamiento es contemplado por:

ElementoDibujable: Es la base fundamental de todo tutorial. Los elementos son objetos que poseen comportamiento propio y pueden ser vistos como una típica caja negra, con una entrada, salida y función de conversión, y que además pueden ser representados en la pizarra donde se

ejecuta el tutorial. En el dominio de la geometría, un objeto típico sería un punto, o una recta. En el dominio de la matemática, un objeto sería, por ejemplo, un número.

Gestores: La creación de un elemento no es una tarea simple ni única, ya que existirán seguramente muchas maneras de hacerlo, por ejemplo, siguiendo con el ejemplo anterior, una recta puede ser creada en base a dos puntos, o puede ser creada en base a un punto y tangente a un círculo, por lo que se necesita de un objeto adicional con el que se asista al usuario en la tarea de reunir los datos necesarios para la creación del elemento, y relacionarlo con el resto del tutorial. Por cada elemento del dominio existirá un gestor, y pueden existir más de uno. En el caso de que el objeto de tipo Elemento no esté pensado para su creación por parte de un usuario, no existirá ningún gestor para el mismo.
El gestor representa las transformaciones necesarias para convertir los eventos de usuario en objetos del dominio, ya sea en el proceso de crear un elemento que sea dato de entrada al tutorial, o en el proceso de crear el elemento resultado, como consecuencia de ejecutar un paso.

2. Estructura del objeto ElementoDibujable

El comportamiento y aspecto de cada elemento de un dominio se comunica con el framework mediante una interfaz común, que cada dominio debe implementar o extender de acuerdo a sus necesidades.

Descripción de la interfaz

Como se observa en la figura 1, la interfaz consta de ocho métodos:

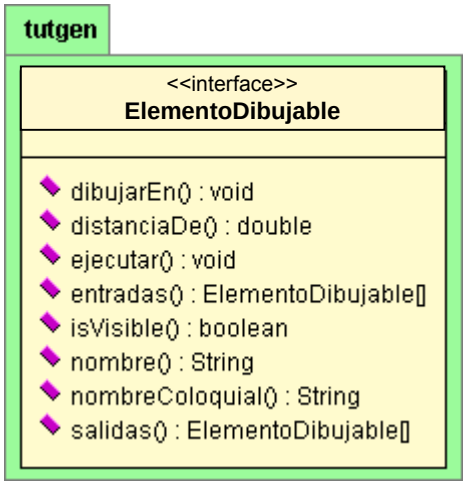

Figura 1 – Diagrama UML de ElementoDibujable

dibujarEn(java.awt.Graphics2D g, , int avance , boolean esFoco, int nivel, int tipo): Este método es llamado por el framework cuando se requiere la visualización del componente. El programador del componente puede implementar el código de visualización y comportamiento en una sola clase o puede delegar la funcionalidad a sus clases según su propio diseño.
TutGen envía como parámetros:

- ✓ **g,** componente gráfico java.awt.Graphics2D en el que debe dibujarse

- ✓ **avance** un valor entero entre 0 y 100 que representa el porcentaje de avance que se requiere de la representación (esto es para provocar efectos de animación cuando el tutorial se está ejecutando paso a paso).
- ✓ **esFoco,** información acerca de si el elemento está enfocado.
- ✓ **nivel,** valor entero no negativo que representa el nivel de anidamiento en que se encuentra dicho componente respecto del tutorial en que forma parte, 0 si es el tutorial principal, 1 si corresponde a un tutorial que es paso del tutorial principal...
- ✓ **tipo,** valor entero que representa si se trata de un componente entrada, intermedio o salida, dado por las constantes TutGen.ENTRADA, TutGen.INTERMEDIO, TutGen.SALIDA.

El nivel, el tipo y el Foco le permite al programador del componente tomar decisiones acerca del nivel de detalles, resalte y efectos especiales que desea dar al componente en cada situación.
Cuando el avance es 100, el objeto debe ser dibujado en forma completa. Cuando la representación de un componente debiera hacerse en cámara lenta, TutGen invoca al método repetidas veces con diferentes valores de avance que van creciendo cada vez, hasta llegar a ser 100. El programador del componente puede utilizar esta información y lograr así el efecto de animación esperado, o ignorarlo y dibujarlo siempre en forma completa.

distanciaDe(double x,double y): Retorna la menor distancia de cualquier parte de su representación gráfica al punto (x, y) cuyas coordenadas se pasan como parámetro.
Es invocado por TutGen cuando necesita identificar qué elementos se encuentran "cercanos" a una determinada posición de la pizarra, o si un elemento en particular se encuentra "cercano" o no a una determinada posición, para dar respuesta a selección, desplazamientos, exploración de propiedades, etc.

- @return double

ejecutar(): señaliza al elemento que debe recalcular sus salidas en base a sus entradas. Es invocado por TutGen cuando requiere un recalculo de los elementos del tutorial, por ejemplo, cuando un objeto ElementoDibujable cambió sus propiedades. El cambio de una propiedad de un elemento puede afectar a otros.
El programador de un dominio podría elegir no implementar esta llamada y actualizar sus salidas cada vez que se requiere su visualización, pero de esta manera se producirían problemas de cálculos innecesarios y no es clara la separación entre código de visualización y ejecución o cálculo.

entradas(): mediante este método el objeto revela parte de sus componentes internos, aquellos que utiliza como datos de entrada. Si es un elemento básico, que no require de otros previamente creados, devuelve null. El framework luego utiliza esta información para almacenar o enlazar tutoriales completos.

- @return ElementoDibujable[] .

salidas(): representa las salidas del objeto producto de su ejecución interna. Devuelve el array de Elementos Dibujables relacionados que lo representan como entidad y que pueden ser referenciados individualmente. Un elemento puede tener muchas salidas o bien ninguna.
Ejemplo: en el caso de un Elemento Recta del dominio Geometría, sus entradas son dos puntos y la salida es la recta, no tiene ninguna otra salida adicional. En un caso más complejo, como podría ser el elemento que representa la intersección entre dos circunferencias, las entradas son dos circunferencias y las salidas pueden ser variables, ya que la intersección será un punto, dos puntos, una circunferencia o bien no existirá si las circunferencias no se interceptan.

- @return ElementoDibujable[]

isVisible(): devuelve un valor lógico que indica si es visible o no. Es invocado por TutGen para decidir si le envia o no el mensaje *dibujarseEn* o *distanciaDe*.

- @return boolean

nombre(): devuelve el nombre del elemento en particular, el cual permite diferenciarlo de otro. TutGen lo invoca cuando requiere comunicar algo al usuario, haciendo referencia a ese elemento en particular.

- @return String

nombreColoquial(): devuelve el nombre coloquial de la clase. Este nombre es utililizado por TutGen para referirse genéricamente a ese tipo de elemento, mientras que para referirse a un elemento en particular, utiliza a *nombre()*. Todos los elementos de una clase tienen el mismo nombre Coloquial.
No tiene por qué ser igual al nombre de su clase, en realidad, debiera ser un nombre significativo para el usuario de TutGen. Además, clases diferentes, podrían tener igual nombre coloquial. Por ejemplo: tanto la clase RectaBean2D como la clase RectaBean3D podrían tener como nombre coloquial a "recta".

- @return String

Esta interfaz representa los requerimientos mínimos de comunicación con el framework.

Utilización del estándar JavaBeans

Aunque no se refleja en la interfaz existe un requerimiento adicional en la estructura de un elemento.
Éste posee propiedades que modifican su comportamiento como por ejemplo transparencia, color, etc. El framework necesita conocer dichas propiedades para hacerlas modificables por el usuario.
Java provee un método estándar para este propósito, llamado JavaBeans que entre otras reglas, especifica la nomenclatura de acceso de las propiedades , cuyo acceso se realiza mediante pares de métodos cuyos nombres comiencen con “set” y “get”. Cada elemento sigue este patrón por lo que se implementa como un bean de java con todas las restricciones y requerimientos adicionales que esto supone (creación de clases beanInfo, etc.), aunque esto no es estrictamente necesario ya que el elemento puede explorarse mediante métodos de introspección y reflexión como se verá en la sección 4.

3. Estructura del objeto Gestor

Como se dijo anteriormente, el objeto Elemento necesita ser “gestionado”, o sea, debe existir un objeto adicional que se encargue de reunir los datos necesarios para la creación del objeto, sea pidiéndoselos al usuario, cargándolos del almacenamiento permanente (disco rígido), o cualquier otra forma. Se diseñó una interface con un comportamiento lo bastante genérico como para servir a la mayor cantidad de dominios posible.

El funcionamiento es el siguiente:

1. Cuando el usuario decide que necesita adicionar un elemento al tutorial, lo selecciona presionando un botón o algún otro control de la interfaz.

2. Esto genera un objeto gestor del tipo correspondiente a ese control, y desde ese momento el framework envía todos los eventos de Mouse o de teclado al gestor, que así obtiene la información necesaria.
3. Una vez reunidos los datos necesarios, el objeto Gestor crea un objeto Elemento, y lo devuelve al framework, que se encargará posteriormente de su representación y ejecución.
4. El gestor es desechado ya que ha finalizado su función, y el framework retoma el manejo de los eventos de Mouse y teclado.

Modo automático

Existe otra manera en la que el framework utiliza los objetos gestores, ya que como se explica en la sección 5, cuando un tutorial es almacenado en el disco, no se almacena estrictamente información acerca de sus elementos, sino los pasos necesarios para la creación del mismo. Por lo tanto en este caso también son necesarios los gestores, aunque actúan sin intervención del usuario, en un modo que se llamará "*automático*".

En este modo el framework simula los eventos producidos por el usuario, alimentándolos al gestor que de esta manera genera el Elemento necesario para recrear el tutorial almacenado.

Descripción de la interfaz

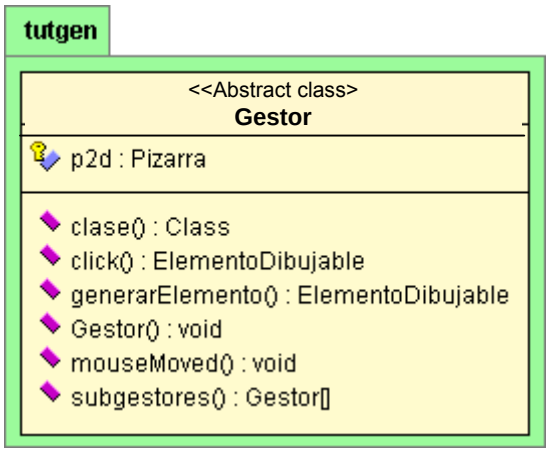

Figura 2 – Diagrama UML de Gestor

clase(): Devuelve la clase de elemento que gestiona. Es solicitada por TutGen para saber la clase de elemento que genera.

- @return Class

click(java.awt.event.MouseEvent e, boolean enModoEntrada): Es invocado cuando el gestor está activo y se produjo el evento mousePressed en la Pizarra. Retorna un nuevo elementoDibujable si es que finaliza su intervención, o null, si aún sigue esperando por más eventos.

TutGen envía como parámetros:

- @param **e:** el evento de mouse que ocasionó el click.
- @param **enModoEntrada:** Información que indica si el tutorial está esperando datos de entrada, cuando es verdadera, o está esperando datos para un paso, cuando es falso.
- @return ElementoDibujable

Cuando el gestor devuelve el elemento, TutGen lo desactiva pues ya finalizó su intervención y retoma el control de eventos. En cambio, mientras el gestor devuelva null, TutGen lo seguirá invovando.

generarElemento (ElementoDibujable datos[]): Devuelve un ElementoDibujable que genera a partir de los elementos que se le pasan como datos de entrada. Es invocado por TutGen cuando requiere gestionar el elemento en modo automático, sin intervención de eventos de usuario.

- @param **datos[]** Es el array de entradas requeridas por el objeto que genera.
- @return ElementoDibujable

Gestor(Pizarra p): El constructor recibe como parámetro a la pizarra en que se despliega el tutorial para el cual está gestando los componentes. Esto es para que el programador del componente pueda tener acceso a la misma, si fuera necesario.

mouseMoved (java.awt.event.MouseEvent e): Es invocado cuando se produce un evento mouseMoved en la Pizarra.

subgestores(): mediante este método el objeto revela parte de sus componentes internos, aquellos que utiliza para gestionar subcomponentes. Si es un Gestor simple, que no require intervención de otros gestores devuelve null. Es requerida para poder tratar en forma uniforme tanto a gestores simples como compuestos.

- @return Gestor[]

Gestor es la cabeza de una jerarquía que cumple con el patrón de diseño Composite [Gamma+95], Gestor es el *Component* y GestorCompuesto es el *Composite*.

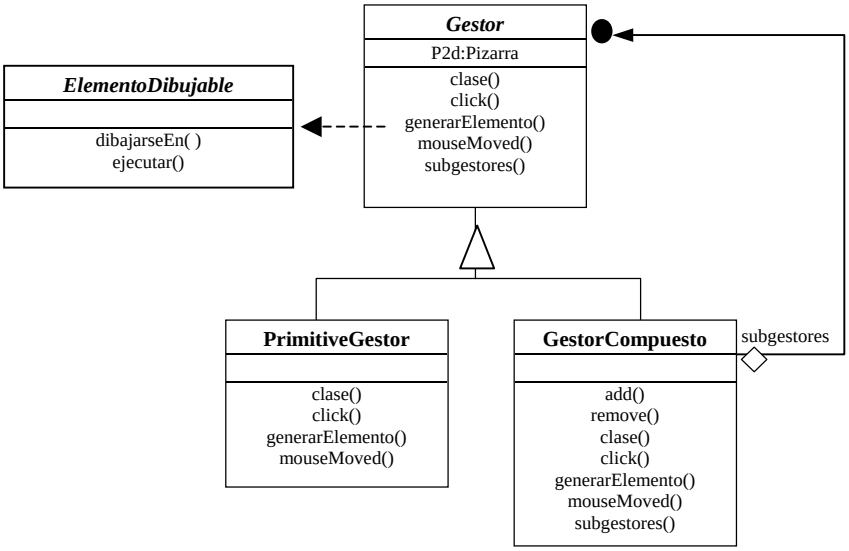

Figura 3 – Estructura del composite Gestor

GestorCompuesto responde a los mensajes, *click*, *mouseMoved* y *generarElemento* retransmitiendo el mensaje a cada uno de sus subgestores. El Elemento devuelto por GestorCompuesto es un CompuestoBean, que contiene a cada uno de los Elementos gestionados por los subgestores.
Una subclase de GestorCompuesto, provista en el framework es GestorTutorialDisco que se encarga de gestionar los tutoriales creados por el usuario para ser utilizados en forma *stand along* o como componente de otro tutorial.

4. Integración al framework

El framework, denominado Tutgen, está diseñado para ser flexible y extensible. Como resultado, desde un principio posee la capacidad de modificar la paleta de componentes con la que se trabaja, agregando o quitando componentes aún en tiempo de ejecución.

El programador que desee implementar componentes para generar tutoriales en algún dominio en particular, deberá implementar los componentes básicos, denominados "*canónicos*":

Elementos básicos del dominio, por ejemplo, en el dominio de geometría serian: punto, recta, circunferencia, intersección, etc.. Estos elementos deben cumplir con el estándar JavaBeans, al menos en la nomenclatura de sus métodos para modificar propiedades e implementar la interface ElementoDibujable.

Gestores para cada elemento, siguiendo el dominio anterior: generar un punto a partir de un clic del Mouse, generar una recta a partir de dos clics del Mouse, generar un punto a partir de la intersección entre dos rectas, etc. El comportamiento para gestores compuestos y de tutoriales almacenados en el disco ya están implementados en el framework.

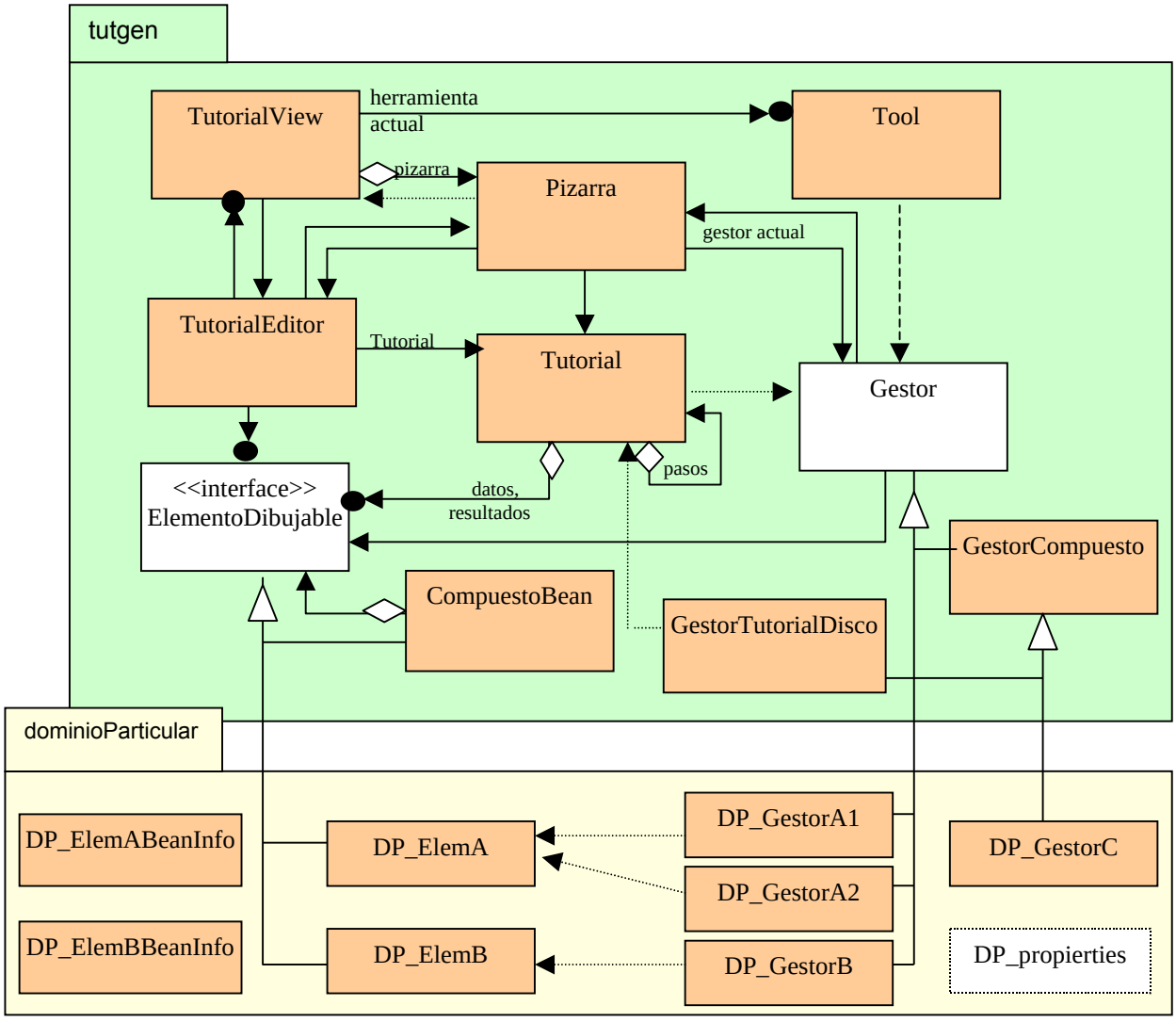

Figura 4 - Diagrama de clases y conexión con los paquetes desarrollados para un dominio.

Un archivo con la lista de gestores disponibles en el dominio. El framework utilizará este archivo para generar la paleta de componentes.

Una clase de tipo BeanInfo: Para que cada elemento del dominio cumpla con el estándar JavaBeans, se debe generar una clase denominada BeanInfo que contiene información acerca del mismo, tal como varios tipos de íconos para su representación, propiedades a publicar, tipo de publicación (sólo lectura o acceso total), nombre de los métodos para la modificación de propiedades, y otras. Sin embargo, Java tiene la posibilidad de ahorrarle al programador la tarea de implementar esta clase adicional, por medio de un mecanismo denominado introspección de clases [Sun2002]. Por medio de este mecanismo, si el sistema JavaBeans no encuentra la respectiva clase BeanInfo correspondiente a un Elemento dado, procederá a inspeccionar la clase del elemento método por método utilizando la característica de reflexión del lenguaje Java, y de esta manera recolectar la información faltante.
De todas maneras es recomendable implementar la clase BeanInfo correspondiente, ya que de esta manera se proveerá mayor información a la hora de determinar las propiedades visibles de un objeto, y se tiene la posibilidad de agregar un ícono para su representación.

Luego de programar estas clases, siguiendo las interfaces definidas por el framework, deben compilarse y colocarse en algún lugar dentro del alcance de la máquina virtual Java.

El paso final es indicar al framework la posición del archivo con la lista de gestores para los componentes canónicos (básicos) del nuevo dominio, tras lo cual se generará automáticamente una paleta de componentes en base a esa información, listos para su utilización.

5. Almacenamiento

Durante la creación del tutorial, se fueron registrando todos los pasos que el autor realizó en un array de objetos denominados 'pasoInteligente'. Estos objetos contienen la información necesaria para reconstruir el tutorial, generando todos los componentes. Básicamente un pasoInteligente contiene un componente, y la descripción del paso y el array de objetos PasosInteligentes contiene toda la información necesaria para recrear un tutorial completo.
Esta información puede almacenarse de distintas maneras, por ejemplo, podría almacenarse utilizando serialización y almacenarse como un *ObjectStream* de Java en el disco, (como se realizaba en las primeras versiones), o podría almacenarse como un archivo con formato propietario, pero la actual versión almacena la información como un archivo XML cuya definición de datos puede verse en el archivo 'tutgen.dtd' en la sección 7, que se utiliza para la validación del tutorial.

Existen dos maneras de leer un archivo XML: mediante un parser DOM o uno SAX. La diferencia estriba en que el parser DOM explora el archivo XML y genera un árbol que luego la aplicación debe leer. Un parser SAX genera eventos en base a un archivo XML que se le alimenta. La aplicación debe proveer funciones que capturen los eventos y así procesar la información.
La actual versión del sistema utiliza el sistema SAX que contiene la especificación Java 1.4, ya que por el momento el framework sólo utiliza la tecnología XML en el momento de grabación y recuperación, por lo que no necesita de un sistema más complejo como DOM.

Una característica muy importante del sistema, es que un tutorial almacenado en disco se convierte en otro componente del dominio listo para ser reutilizado en la creación de un tutorial mas complejo

[Rosanigo+02b], por lo que un tutorial puede estructurarse en capas, cada una de las cuales posee mayor detalle que la anterior, como se detalla en la figura siguiente.

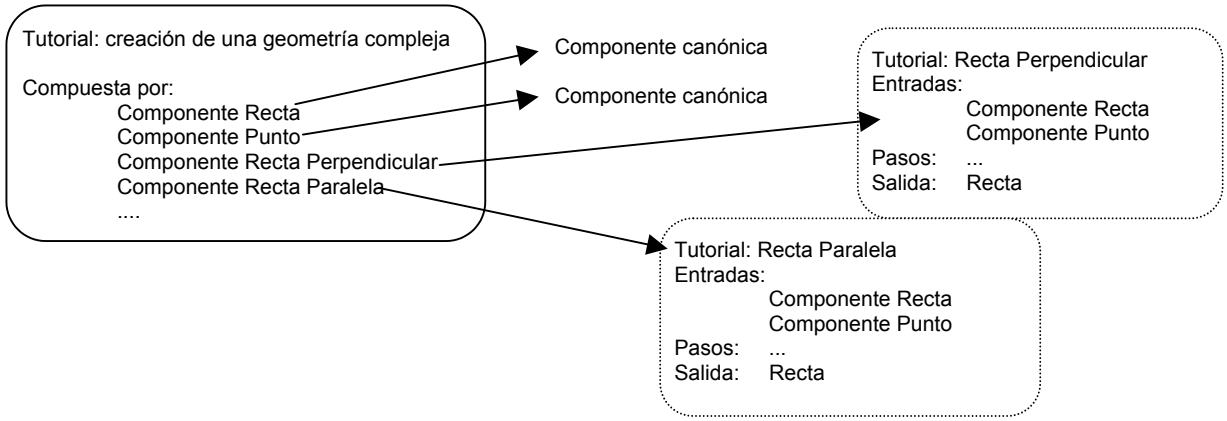

Figura 5 – Ejemplo de Tutorial compuesto

6. Conclusiones y futuros trabajos

Aunque el proyecto en el cual se utiliza el software descripto no finalizó a la fecha de creación de este artículo, y el sistema no ha sido utilizado a gran escala, podemos afirmar:

- ✓ La manera de estructurar los componentes mencionados promete una gran flexibilidad y facilidad de uso, como lo está demostrando en las implementaciones realizadas dentro del grupo de investigación informática local.
- ✓ El desarrollador de componentes para un dominio particular, debe cumplir con una pequeña interfaz impuesta por el framework, y tiene total libertad para caracterizar a cada elemento del dominio y puede dejar fácilmente, valores de propiedades editables por el usuario de la aplicación.
- ✓ El framework permite crear de manera sencilla tutoriales en cualquier dominio del conocimiento. Simplemente requiere contar con los componentes que representan los datos y acciones primitivas en esa área del conocimiento.

Continuando con esta línea de investigación y desarrollo, nuestro próximo trabajo está orientado a complementar al generador de tutoriales con entrenador genérico e inteligente, que interprete de cada tutorial los diferentes caminos que conducen a la solución, los reconozca como válidos sin necesidad de que el docente deba expresarlos uno por uno, y permita asistir al alumno en sus errores guiándolo hacia la solución correcta, sin necesidad de la permanente presencia del docente, a la vez que le brinda un mecanismo de autoevaluación.

7. Definición de datos de un tutorial (DTD)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT Tutorial (Titulo, Autor, Descript, Version, Entradas, Componentes) >
<!ELEMENT Titulo ( \#PCDATA ) >
<!ELEMENT Autor ( \#PCDATA ) >
<!ELEMENT Descript ( \#PCDATA ) >
<!ELEMENT Version ( \#PCDATA ) >
<!ELEMENT NombreGestor ( \#PCDATA ) >
<!ELEMENT Propiedades ( Propiedad+ ) >
<!ELEMENT Id EMPTY >
<!ATTLIST Id Valor NMTOKEN \#REQUIRED >
<!ELEMENT Gestor ( Id, NombreGestor, IdSubComponentes, Propiedades? ) >
<!ELEMENT IdSubComponentes ( Id+ ) >
<!ELEMENT Entradas ( Gestor ) >
<!ELEMENT Salidas ( Gestor ) >
<!ELEMENT Propiedad EMPTY >
<!ATTLIST Propiedad Nombre NMTOKEN \#REQUIRED >
<!ATTLIST Propiedad Valor CDATA \#REQUIRED >
```

8. Glosario

XML (eXtensible Markup Language) Estándar impulsado por el W3C (WorldWideWeb Consortium) para almacenar información estructurada en formato de texto.
parser: Código utilizado para leer un documento y analizar su estructura. En este artículo, se utiliza este término para nombrar al componente utilizado para procesar archivos XML.
DOM (Document Object Model) Parser recomendado de manejar archivos XML, en el que se debe manejar una estructura de datos tipo árbol.
SAX: (Simple API for XML) Estándar defacto para la utilización de archivos XML, en el que se procesan los datos mediante una serie de eventos.
IDE: (Integrated Develpment Enviroment) Entorno integrado de desarrollo, suele ser una aplicación que permite editar, generar y proveer asistencia para la creación de aplicaciones de algún tipo.

Referencias

[Booch+98] BOOCH G., JACOBSON I., RUMBAUGH J.*The Unified Process Sofwtare Development* Addison-Wesley Publications, 1998

[Cooper98] COOPER, JAMES W. - *Java Design Patterns: A Tutorial*, 1998 – Addison Wesley

[Eckel-00] Eckel, Bruce. *Thinking in Java* 2nd ed. ISBN 0-13-027363-5- Prentice Hall

[Gamma+95] GAMMA, ERIC; HELM, RICHARD; JOHNSON, RALPH AND VLISSIDES, JOHN, *Design Patterns. Elements of Reusable Software*, Addison-Wesley, 1995

[PI+2001] ROSANIGO ZULEMA B.; BRAMATI PEDRO, PAUR ALICIA; ORTEGA ALFREDO; CERRA JOSÉ P – *Construcción de tutoriales basados en componentes reusables* Proyecto de investigación 383 – Facultad de Ingeniería – U.N.P.S.J.B.- 2001

[Pressman 97] R. PRESSMAN *Ingeniería del Software, un Enfoque Práctico*, 4º Ed., Mc. Graw Hill. 1997.

[Rosanigo+02a] ROSANIGO ZULEMA B.; BRAMATI PEDRO, PAUR ALICIA; ORTEGA ALFREDO; CERRA JOSÉ P. *Software Educativo basado en Componentes Reusables* Resúmenes en Actas de Congresos ¨III Congreso Nacional de Expresión Gráfica y arquitectura ¨ Facultad de Ingeniería- Universidad Nac. Del Centro de la Provincia de Buenos Aires- Olavarría - Octubre 2002

[Rosanigo+02b] ROSANIGO ZULEMA B.; PAUR ALICIA; BRAMATI PEDRO; ORTEGA ALFREDO; CERRA JOSÉ P.*Un framework para generación de tutoriales*. Actas VIII Congreso Argentino de Ciencias de la Computación (CACIC 2002) Buenos Aires.

[Sun2002] Java™ 2 Platform, Standard Edition, v 1.4.0 API Specification - Copyright 1993-2002 Sun Microsystems, Inc. 901 San Antonio Road