Análisis y Diseño de Ambientes de Modelado y Simulación

UTILIZANDO EL PARADIGMA DE PATRONES

Jaime Muñoz Arteaga [1], Roberto Carrera M. [2], Gustavo Rodríguez G. [3] y Héctor Pérez G. [4]

INTRODUCCIÓN

Los Ambientes de Modelado y Simulación (AMS) son aplicaciones interactivas que permiten el diseño y la simulación de modelos. Los modelos son abstracciones de la realidad (tal como la representación de un sistema del mundo real) que, sin entrar en detalle, permiten razonar y evaluar dicha abstracción. El mayor beneficio que el usuario puede obtener de un ambiente de simulación es la simulación del modelo mismo, ya que puede experimentar y comprender el comportamiento del sistema modelado con menos tiempo, costo y riesgo con respecto al sistema real.

La construcción de AMS es reconocido como una tarea difícil por parte de los trabajos en el área de la Interacción Humano-Computadora (IHC) y la Ingeniería de Software (IS), en los cuales se han identificado dos grandes problemáticas (Rodríguez et al. 2004):

- La dificultad para construir el software de un AMS con un alto nivel de utilidad y de usabilidad.
- Los problemas de comunicación entre los especialistas involucrados en el desarrollo de un AMS.

El ofrecer un alto nivel de funcionalidad del software de un AMS es primordial para asegurar un alta eficiencia y precisión en los modelos a simular. Al mismo tiempo, se requiere de un alto nivel de usabilidad en la interfaz gráfica; es decir, que la interfaz facilite y guíe la tarea del usuario. Esto último también es importante, ya que no es raro conseguir AMS eficientes con interfaces que inciden en una baja usabilidad.

El cuanto al segundo aspecto, se refiere al carácter pluridisciplinario de los equipos de desarrollo de un AMS. Un equipo de desarrollo de software comprende a los especialistas en informática, a los especialistas del dominio y, posiblemente, especialistas de otras disciplinas, así como al usuario. Desafortunadamente, la falta de una metodología que facilite la fluidez de la comunicación entre las partes afecta en el costo, tiempo y esfuerzo del desarrollo de un AMS, lo que provoca que el desarrollo del AMS se realice frecuentemente de manera independiente y se desarrollen desde cero, mecanismos ya existentes, en particular aquellos que se refieren a la interfaz de usuario y/o la funcionalidad de un AMS.

Con el fin de mitigar la problemática anterior, el presente trabajo propone una metodología de análisis y diseño de ambientes de modelado y simulación, basándose en el paradigma de patrones, los cuales permiten especificar este tipo de ambientes, independientemente de un lenguaje de programación, y a la vez facilitan la

[1] Centro de Ciencias Básicas, Departamento de Sistemas de Información de la UAA. jmunozar@correo.uaa.mx
[2] Ingeniería en Computación, Unidad Académica de Ingeniería Eléctrica. Universidad Autónoma de Zacatecas (UAZ). rcarrera@ieee.org
[3] Coordinación de Ciencias Computacionales del Instituto Nacional de Astrofísica Optica y Electrónica (INAOE). jrodrig@inaoep.mx
[4] Facultad de Ingeniería de la Universidad de San Luis Potosí (UASLP) hectorgerardo@acm.org

comunicación con el personal involucrado en el desarrollo de software.

MATERIALES Y MÉTODOS

Por definición, un patrón es una solución a un problema recurrente dentro de un contexto específico (Gamma *et al.* 1995). La especificación de un patrón permite comunicar la experiencia y el conocimiento en un área de interés. A partir de los noventa, y hasta nuestros días, una gran diversidad de patrones se han propuesto, entre los que destacan los patrones de cómputo científico (Rodríguez *et al.* 2004), los patrones de interacción (Welie and Trætteberg, 2000); (Muñoz *et al.* 2004) y los patrones de diseño de software (Gamma *et al.* 1995). Todo tipo de patrones tienen un formato bien definido donde se especifica de manera general: el nombre del patrón, problema que resuelve, solución propuesta, contexto y ejemplo.

Ahora bien, los patrones en el contexto del desarrollo de ambientes de simulación son un medio para registrar las experiencias exitosas durante sus diferentes etapas de concepción. El uso de patrones conlleva una serie de beneficios tales como la reutilización, alto nivel de abstracción y un fácil mantenimiento de los componentes de un AMS. Razón por la cual, el presente trabajo propone especificar la funcionalidad y la usabilidad de un AMS respectivamente en términos de patrones de diseño y de patrones de interacción.

- Los patrones de diseño permiten diseñar el código de la parte no interactiva de un sistema, cumpliendo criterios de calidad del software orientado a objetos, como por ejemplo "el favorecer la composición sobre la herencia" y "diseñar para el cambio" (Gamma *et al.* 1995).

- Los patrones de interacción especifican las buenas prácticas para diseñar la interfaz del usuario, basándose en los requerimientos del usuario (Muñoz *et al.* 2004) y tomando en cuenta criterios de satisfacción de éste cuando utiliza una aplicación interactiva, como por ejemplo la facilidad de uso, la asistencia en caso de errores, la facilidad de aprender y el tiempo de respuesta de la aplicación.

La metodología para diseñar un AMS, en términos de patrones de interacción y de diseño, comienza con un análisis previo de los requerimientos del usuario y del sistema. Conforme a la figura anterior, esta metodología se conforme de seis etapas, a saber:

1. Analizar los requerimientos del usuario, de manera que especifique las preferencias y la tarea de éste.
2. Especificar los requerimientos del sistema a través de casos de uso, de manera que describa el conjunto de servicios que ofrece el sistema al usuario.
3. A partir de los dos tipos de requerimientos anteriores, proponer un prototipo de interfaz gráfica.

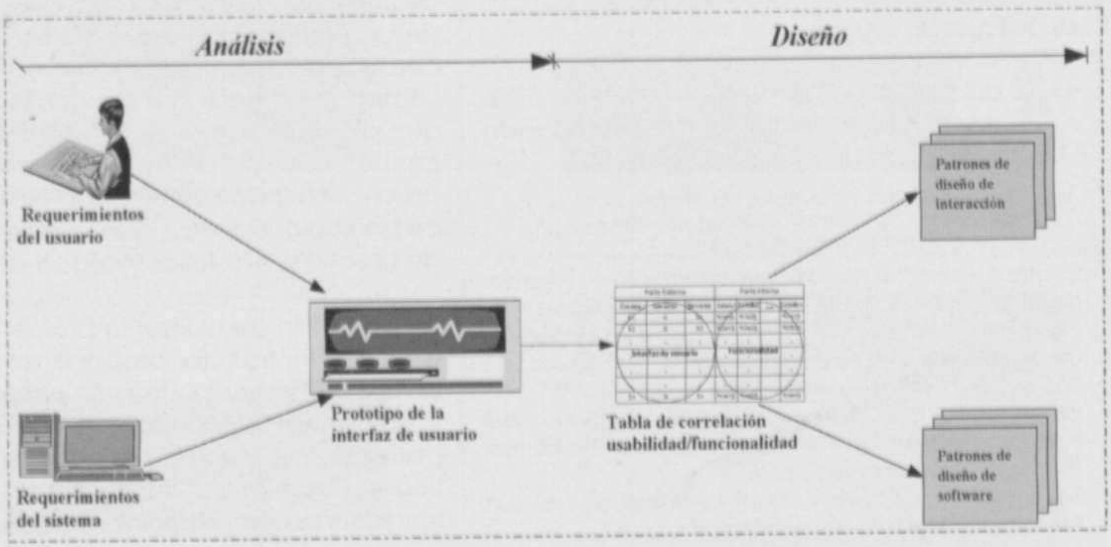


Figura 1.- Análisis y Diseño de Ambientes de Modelado y Simulación Utilizando el Paradigma de Patrones.

4. Asociar, por cada interactor de la interfaz gráfica, la acción del usuario, el servicio (o método) que se ejecutará y el estado de interacción. Estos dos últimos elementos se encapsulan en una clase orientada a objetos. El resultado de la asociación de cada uno de los interactores de la interfaz gráfica conforma una tabla de correlación de elementos de Usabilidad/ Funcionalidad.
5. Seleccionar los patrones de interacción del catálogo propuesta por Muñoz (Muñoz *et al.* 2004) a partir de los componentes de usabilidad (interactor y acciones del usuario) determinados en la tabla de correlación de la etapa anterior.
6. Seleccionar los patrones de diseño del catálogo propuesta por Gamma (Gamma *et al.* 1995) a partir de los componentes de funcionalidad (servicio, estado de interacción y clase orientado a objetos) especificados en la tabla de correlación de la etapa cuatro.

RESULTADOS

Con el fin de mostrar la aplicación de la metodología propuesta en la sección anterior, se presenta aquí el análisis y diseño de un ambiente simulador de señales.

Requerimientos del usuario y del sistema.

El usuario requiere de un ambiente de simulación de señales donde él pueda generar señales y visualizarlas a través de diversos tipos de filtros y tratamientos. El usuario necesita comparar al menos dos señales de entrada, que pueden variar de onda generadora; la frecuencia deberá ser definida al momento de ejecución, así como el filtro y tratamiento que recibirá la señal. Tanto las señales generadas, como los procesamientos recibidos se deberán observar de forma gráfica durante el tiempo de la simulación.

Los requerimientos del sistema que debe cumplir el ambiente de simulación de señales son plasmados en el caso de uso de la figura 2, donde en la primera columna están descritos los servicios que el usuario tiene acceso a través de la interfaz gráfica.

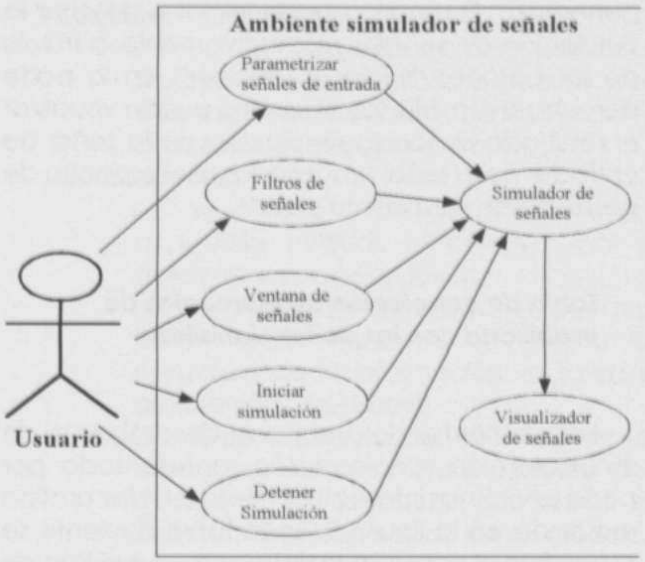


Figura 2.- Diagrama de de casos de estudio para el ambiente de simulación de señales.

Prototipo de la interfaz de usuario.

A partir de los requerimientos del usuario y del sistema, se propone el siguiente prototipo de la interfaz de usuario:

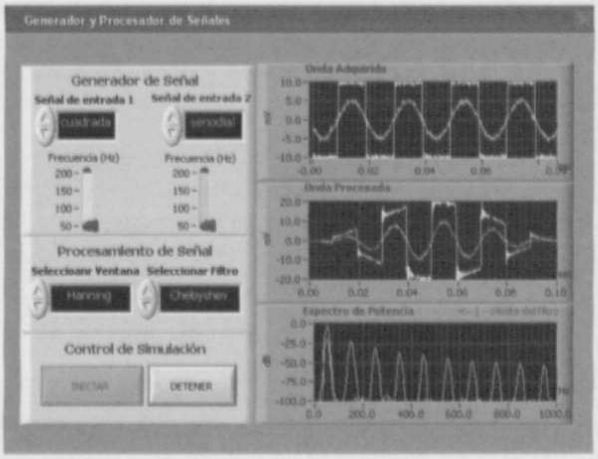


Figura 3.- Interfaz de usuario del ambiente de simulación de señales.

En la interfaz de usuario del simulador de la figura 3, la parte izquierda está dedicada al control de las señales generadas; la parte derecha corresponde a la visualización de dichas señales. El panel de control "Generador de Señal" permite al usuario seleccionar la frecuencia y fijar el tipo de señal de entrada (v.gr. senoidal, cuadrada y triangular). El panel de control "Procesamiento de Señal", permite seleccionar el tipo de ventana y filtro que se puede aplicar a las señales

generadas. El usuario puede iniciar o detener la simulación de señales respectivamente a través de los botones "Iniciar y Detener". En la parte derecha de la interfaz el usuario puede visualizar el resultado de cada simulación de la señal de entrada, de la señal procesada y del espectro de potencia de esta última señal.

Tabla de correlación de elementos de usabilidad con los de funcionalidad

Los elementos de usabilidad del ambiente de simulación de señales están representada por cada uno de los interactores de la interfaz gráfica mostrada en la figura 3. En la tabla siguiente se especifica el conjunto de interactores y el tipo de evento a los cuales responde cada uno ellos de esta interfaz gráfica.

La funcionalidad de cada interactor está determinada por una clase orientada a objetos asociada a través del servicio (o método) que lo implementa que, al ejecutarse, puede cambiar la variable de estado de dicho objeto. Por ejemplo, cuando el usuario hace "Click" con el cursor del ratón sobre el interactor "Botón Iniciar Simulación" ejecuta el servicio Start Simulation, el cual permite la simulación de generación y procesamiento de señales.

La forma tabular permite establecer "el qué hace" el AMS, poniendo en relación de los elementos de usabilidad con los de funcionalidad del ambiente de simulación. En las dos subsecciones siguientes se presenta "el como hacer" el diseño e la interfaz de usuario especificando particularmente la tarea del usuario y las relaciones entres las clases orientadas a objetos.

Especificación de la interfaz en términos de patrones de interacción

A partir de la información obtenida de la tabla de correlación Usabilidad/Funcionalidad y de las clasificaciones de patrones de interacción de Van Welie (Welie and Trætteberg, 2000) y de Muñoz (Muñoz *et al.* 2004), se identificaron 2 patrones de interacción donde se aplican prácticas exitosa en el diseño de la interfaz de usuario del simulador de señales.

Nombre	**Gestión de servicios**

Problema
El usuario necesita conocer los servicios u operaciones válidas para manipular el modelo a simular.

Principio de usabilidad
Predictibilidad y acciones explícitas.

Contexto
El espacio de interacción debe ser visualizado continuamente por el usuario.

Fuerza
- Guiar al usuario en sus acciones, ofreciendo los servicios disponibles en función del estado actual de una aplicación interactiva.

Elementos de usabilidad		Elementos de funcionalidad		
Interactor	**Eventos**	**Servicios**	**Estado**	**Clase**
Caja de Edición Señal de Entrada 1	Select	Set Signal (1)	Edición	Edit Command
Caja de Edición Señal de Entrada 2	Select	Set Signal (2)	Edición	Edit Command
Selector Frecuencia 1	Drag	Set Frecuency()	Edición	Signal
Selector Frecuencia 2	Drag	Set Frecuency ()	Edición	Signal
Caja de Edición Ventana	Select	Set Window ()	Edición	Signal
Caja de Edición Filtro	Select	Set Filter ()	Edición	Signal
Botón Iniciar Simulación	Click	Start Simulation ()	Activación	Start Command
Botón Detener Simulación	Click	Stop Simulation ()	Activación	Stop Command
Gráfica 1 None Event	Plot ()	Graficación	Plot	
Gráfica 2 None Event	Plot ()	Graficación	Plot	
Gráfica 3 None Event	Plot ()	Graficación	Plot	

Tabla 1. Tabla de correlación Usabilidad/Funcionalidad.

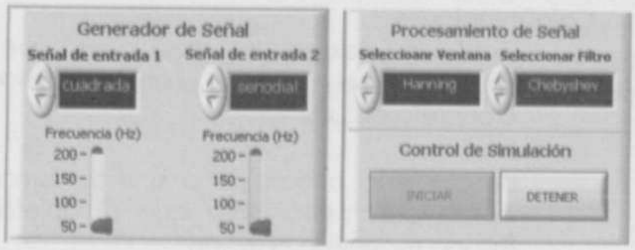
Solución

Para indicar la disponibilidad de un servicio al usuario se deben resaltar los objetos de interacción asociado al servicio y atenuarlos cuando el servicio llega a estar indisponible.

Consecuencia

El usuario tiene el control sobre la interacción y puede decidir fácilmente su siguiente acción.

Ejemplo



El conjunto de servicios que el usuario puede realizar en el ambiente de simulación es el conjunto de interactores de edición para fijar los tratamientos de señal, así como los botones "Iniciar y Detener" para tener el control de la simulación.

Nombre	Visualización de las variables de estado

Problema

¿Cómo facilitar la interpretación y visualización de las variables de estado durante la simulación del modelo de interés?

Principio de usabilidad

Reducirlacargacognitivayrepresentación coherente.

Contexto

El sistema debe de visualizar su estado conforme cambia en el tiempo, en particular cuando el estado está representado por numerosas variables.

Fuerzas

Mostrar todo cambio de esta-do del sistema de manera ordenada.

• La interfaz debe permitir interactuar al usuario con la información.

Solución

Uso de metáforas visuales para facilitar al usuario la interpretación, el acceso y el despliegue de los valores de las variables de estado del modelo a simular.

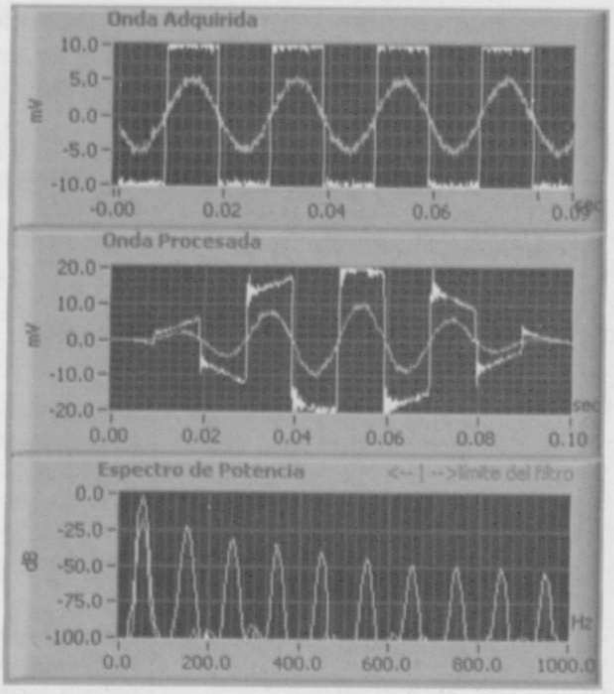
Consecuencia

La visualización del estado permite al usuario evaluar su objetivo final y determinar su acción futura.

Las opciones de visualización pueden personalizarse tomando en cuenta la estructura de la información, la tarea y preferencias del usuario.

Dicha representación también le permite tener una imagen mental correcta del acceso de la información que ofrece el sistema a un estado dado de un modelo.

Ejemplo



La imagen visualiza el resultado de la simulación de señal de entrada (gráfica superior) y de la señal procesada (gráfica intermedia), la gráfica inferior muestra el espectro de potencia de las señales procesadas.

Especificación de la interfaz en términos de patrones de software.

A partir de la información obtenida de la tabla de correlación Usabilidad/Funcionalidad y de la clasificación de patrones de diseño propuesto

(Gama *et al.*1995), se identificaron tres patrones para aplicar las buenas prácticas para el diseño del estado, la vista y los servicios que ofrece en un momento de la interacción el ambiente de simulación de señales.

Nombre **Estado**

Propósito

Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Parecerá que cambia la clase del objeto.

Motivación

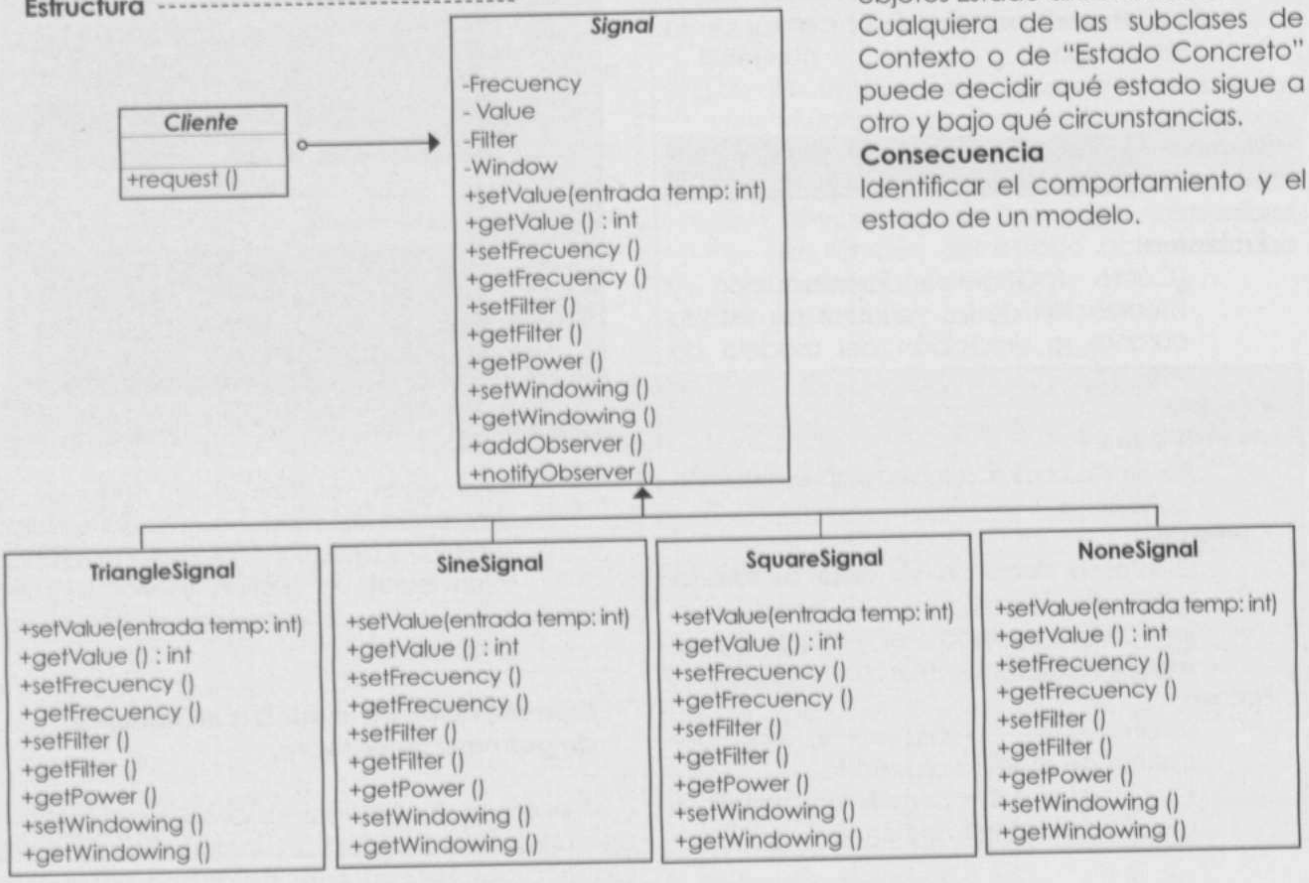
Poder mantener y conocer el estado de un objeto.

Uso

Este patrón se puede usar en cualquiera de estos dos casos:

- El comportamiento de un objeto depende de su estado, y debe de cambiar en tiempo de ejecución dependiendo de ese estado.
- Las operaciones tienen largas sentencias condicionales con múltiples ramas que dependen del estado del objeto.

Estructura



Participantes

Contexto "(Client)": Define la interfaz de interés para los clientes. Y mantiene una instancia de una subclase de Estado Concreto que define el estado actual.

Estado "(Signal)": Define una interfaz para encapsular el comportamiento asociado con un determinado estado del Contexto.

Subclases de Estado Concreto (Triangle, Sine, Square, None-Signal): Cada subclase implementa un comportamiento asociado con un estado del Contexto.

Colaboraciones

Contexto delega las peticiones que dependen del estado en el objeto Estado Concreto actual.

Un contexto puede pasarse a sí mismo como parámetro para que el objeto Estado maneje la petición.

Contexto es la interfaz principal para los clientes. Los clientes pueden configurar un contexto con objetos Estado. Una vez que está configurado el contexto, sus clientes ya no tienen que tratar con los objetos Estado directamente.

Cualquiera de las subclases de Contexto o de "Estado Concreto" puede decidir qué estado sigue a otro y bajo qué circunstancias.

Consecuencia

Identificar el comportamiento y el estado de un modelo.

Nombre Observador

Propósito

Define una dependencia de uno a muchos entre objetos, de forma que, cuando un objeto cambie de estado, se notifique y se actualicen automáticamente todos los objetos que dependen de él.

Motivación

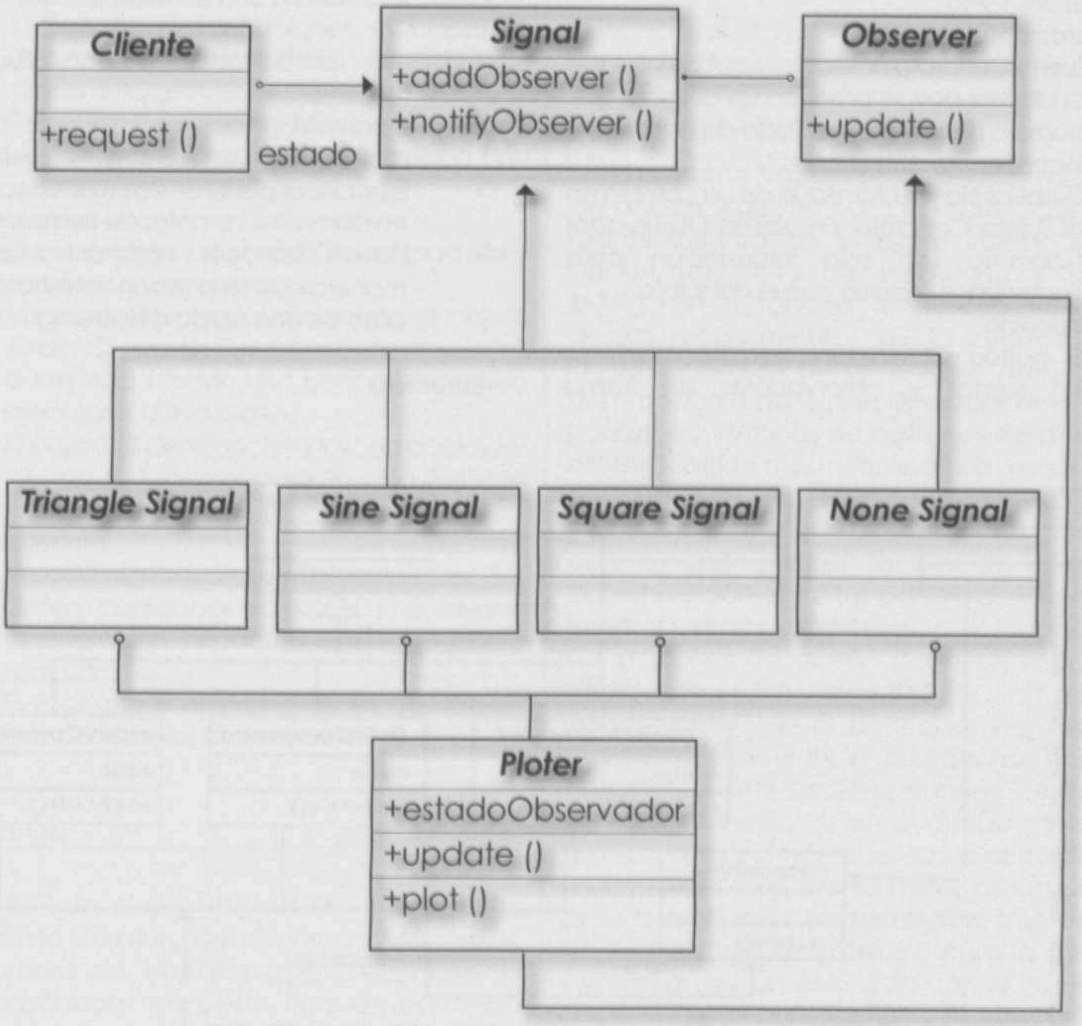
Un efecto lateral habitual de dividir un sistema en una colección de clases cooperantes es la necesidad de mantener una consistencia entre objetos relacionados. No se quiere alcanzar esa consistencia haciendo a las clases fuertemente acopladas, ya que esto reducirá su reutilización.

Uso

Aplíquese este patrón en cualquiera de las siguientes situaciones:

- Cuando una abstracción tiene dos aspectos y uno depende del otro. Encapsular estos aspectos en objetos separados permite modificarlos y reutilizarlos de forma independiente.
- Cuando un cambio en un objeto requiere cambiar otros, y no se sabe cuántos objetos se requieren cambiar.
- Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quiénes son dichos objetos. En otras palabras, cuando no se quiere que estos objetos estén fuertemente acoplados.

Estructura



Participantes

Sujeto ("Signal"): conoce a sus observadores. Un sujeto puede ser observado por cualquier número de objetos Observador. Proporciona una interfaz para asignar y quitar objetos Observador.

Observado ("Observer"): define una interfaz para actualizar los objetos que deben ser notificados ante cambios en un sujeto.

SujetoConcreto ("Triangle", "Sine", "Square", "None-Signal"): almacena el estado de interés para los objetos Observador Concreto. Además envía una notificación a sus observadores cuando cambia su estado.

Observador Concreto ("Ploter"): Mantiene una referencia a un objeto Sujeto Concreto; guarda un estado que debería ser consistente con el del sujeto; implementa la interfaz de actualización del Observador para mantener su estado consistente con el del sujeto.

Colaboraciones

Sujeto Concreto notifica a sus observadores cada vez que se produce un cambio que pudiera hacer que el estado de éstos fuera inconsistente con el suyo.

Después de ser informado de un cambio en el Sujeto Concreto, un objeto Observador Concreto usa esta información para sincronizar su estado con el del sujeto.

Consecuencia

El patrón Observador permite modificar los sujetos y observadores de forma independiente. Es posible reutilizar objetos sin reutilizar sus observadores y viceversa. Esto permite añadir observadores sin modificar el sujeto u otros o observadores.

Nombre: Comando

Propósito

Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones, hacer fila, llevar un registro de las peticiones o poder deshacer las operaciones.

Motivación

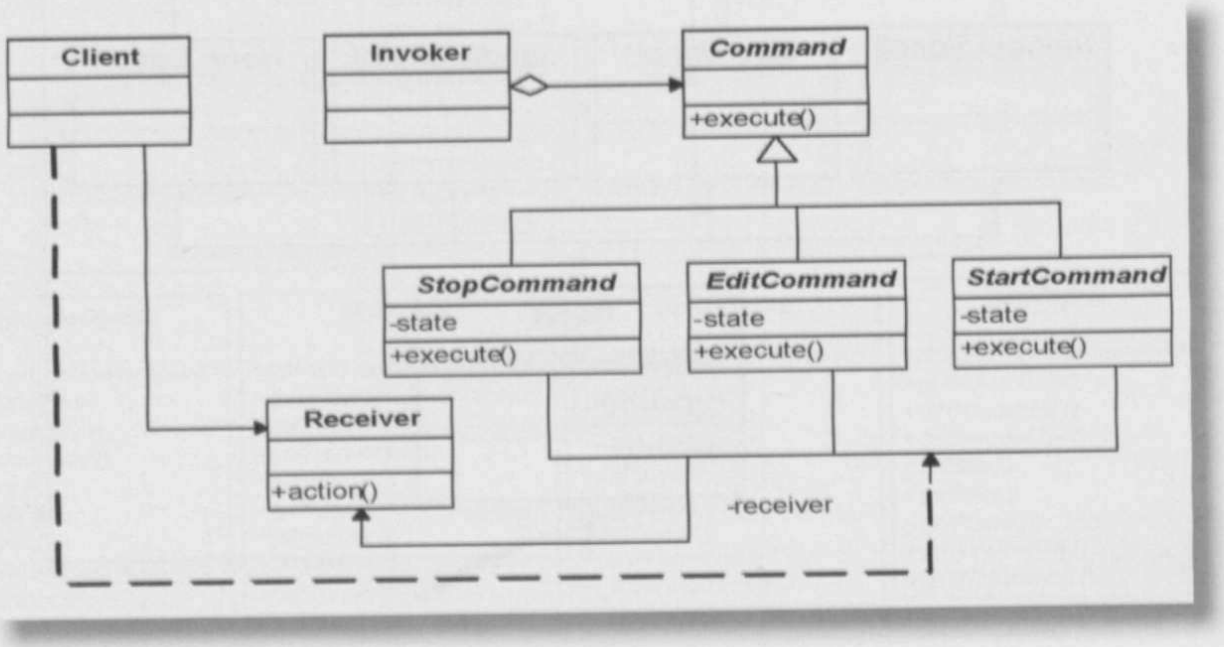
Cuando es necesario enviar peticiones a objetos sin saber acerca de la operación solicitada o de quién es el recetor de la petición. Por ejemplo los "toolkits" de interfaces de usuario incluyen botones y menús que realizan una petición en respuesta a una entrada de usuario.

Uso

Usar el patrón comando cuando se requiera:

- Parametrizar objetos con una acción a realizar como ocurre con los objetos.
- Especificar, poner en fila y ejecutar peticiones en diferentes instantes de tiempo.
- Permitir deshacer y registrar los cambios de manera que se puedan volver a aplicar en caso de una caída del sistema.

Estructura



Trabajos/Patrones de diseño de	(Kreutzer, 1996)	(Blilie, 2002)	(Sanz et al. 2003)	(Ekström 2001)	(Boyko et al. 2002)	InOutSpec
Interfaz					*	*
Software	*	*	*	*	*	*

Tabla 2. Comparación de trabajos de ambientes de modelado y simulación en base en el paradigma de patrones.

Participantes

Orden ("Command") : declara una interfaz para ejecutar una operación.

OrdenConcreta ("StopCommand", "EditCommand", "StartCommand"): define un enlace entre un objeto Receptor y una acción, implementa el método de ejecución.

Cliente ("aplicación"): Crear un objeto OrdenConcreta y establece su receptor.

Invocador ("InputSignalCombobox/2", "startbutton", "stopButton"): Le pide a la orden que ejecute la petición.

Receptor ("aplicación"): Sabe cómo llevar a cabo las operaciones asociadas a una petición.

Colaboraciones

El cliente crea un objeto OrdenConcreta y especifica su receptor.

Un objeto Invocador almacena el objeto OrdenConcreta.

El Invocador envía una petición llamando ejecutar sobre la orden.

El objeto OrdenConcreta invoca operaciones de su receptor para llevar a cabo la petición.

Consecuencia

He aquí algunas de las consecuencias:

1. Orden desacopla el objeto que invoca la operación de aquél que sabe como realizarla.
2. Es fácil añadir nuevas ordenes, ya que no hay cambiar las clases existentes.

DISCUSIÓN

A pesar que el paradigma de patrones es reciente en la literatura del diseño de AMS, existe cierto número de trabajos que preconizan su diseño, basándose en cierto tipo de patrones. Los tipos de patrones más frecuentes, tal como lo muestra la siguiente tabla, son los patrones de interacción y los patrones de software.

El trabajo aquí propuesto titulado In&OutSpec (Especificación de la parte interna y externa de una aplicación interactiva) aparece en la tabla 2 como el único que propone especificar la interfaz de usuario y la funcionalidad de un AMS, utilizando respectivamente los patrones de interacción y los patrones de diseño de software.

CONCLUSIONES

El presente trabajo propone una metodología de análisis y diseño para la reutilización y el fácil mantenimiento de la parte interna y externa de un Ambiente de Modelado y Simulación (AMS). En el análisis establece una relación de los requerimientos de la interfaz gráfica y la funcionalidad de un AMS. A nivel de diseño se propone una especificación más cerca al código en términos de patrones de diseño y una especificación más cercana a los requerimientos del usuario en términos de patrones de interacción. Uno de los objetivos alcanzados con los patrones es facilitar la fluidez de la comunicación entre las partes, donde cada una tiene su propia experiencia y conocimientos. Los patrones de diseño permiten especificar el código de la funcionalidad de una aplicación, dando soluciones al diseñador para reutilizar y mantener fácilmente el código. Por su parte, los patrones de interacción permiten especificar las experiencias exitosas en el diseño de la interfaz gráfica, tomando en cuenta las necesidades del usuario y los componentes gráficos. El modelo aquí propuesto puede ser extendido a la aplicación de los patrones de software para la generación de código de un AMS. Esto con el fin de llevar a cabo el prototipaje rápido y realizar así pruebas en conjunto con el usuario.

REFERENCIAS

- Blilie, C.: *Computing in Science & Engineering Patterns in scientific software: an introduction* (2002).
- Ekström, U.: *Design Patterns for Simulations in Erlang/OTP*. PhD Thesis, Uppsala University, Sweden (2001).
- Gamma, E., Helm, R., Johnson, R., Vlissides, J. and Booch, G.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing (1995).
- Kreutzer, Wolfgang: *Some Patterns for Building Discrete Event Models and Simulators*. Urbana Illinois., USA (1996).
- Muñoz, A.J., Reyes, G.C.A. and Pérez, G.H.G.: Designing Direct Manipulation User Interfaces by Using Interaction Patterns. *WSEAS Transactions on Computer 3* (2004).
- Rodríguez, G.G., Muñoz, A.J. and Fernández, d.B.R.: *Scientific Software Design Through Scientific Computing Patterns. 4th IASTED International Conference on Modeling, Simulation, and Optimization -MSO04-* (2004) 493-498.
- Sanz, R. and Zalewski, J.: "Pattern-Based Control Systems Engineering" in *IEEE Control System* Vol. 23, No. 3, (2003) 43-60.
- Peter Forbrig, Q.L.B.U.&.J.V., (ed.): "User Interface Desing Patterns for Interactive Modeling" in *Demography and Biostatics. Rostock, Germany: Springer-Verlag*. (2002).
- Welie, van Martijn and Trætteberg, Hallvard. "Interaction Patterns" in *User Interfaces, 7th. Pattern Languages of Programs Conference*; Allerton Park Monticello, Illinois, USA. (2000).