(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 599 006**

(51) Int. CI.:

G06F 9/54 (2006.01)
G06F 17/50 (2006.01)
G06F 9/44 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **06.01.2005 PCT/US2005/000390**

(87) Fecha y número de publicación internacional: **04.08.2005 WO05071537**

(96) Fecha de presentación y número de la solicitud europea: **06.01.2005 E 05705164 (1)**

(97) Fecha y número de publicación de la concesión europea: **24.08.2016 EP 1706818**

(54) Título: **Sistema y procedimiento de programación de la ejecución de componentes de un modelo usando eventos del modelo**

(30) Prioridad:

15.01.2004 US 759346

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
31.01.2017

(73) Titular/es:

THE MATHWORKS, INC. (100.0%)
3 APPLE HILL DRIVE
NATICK, MA 01760, US

(72) Inventor/es:

SZPAK, PETER y
ENGLEHART, MATTHEW

(74) Agente/Representante:

CARPINTERO LÓPEZ, Mario

DESCRIPCIÓN

Sistema y procedimiento de programación de la ejecución de componentes de un modelo usando eventos del modelo

La realización ilustrativa de la presente invención se refiere en general a la ejecución de los componentes del modelo dentro de un entorno de modelado, y más específicamente a la programación de la ejecución de los componentes de modelado usando los eventos del modelo.

La realización ilustrativa de la presente invención está relacionada con una solicitud de patente de Estados Unidos actualmente pendiente titulada, "*A System and Method for Using Execution Contexts in Block Diagram Modeling*", número de serie: 10/414, 644.

Simulink™ de The MathWorks, Inc. de Natick, Massachusetts, es un ejemplo de un entorno de modelado gráfico, específicamente un entorno de diagrama de bloques. Simulink™ permite a los usuarios crear un modelo pictórico de un sistema dinámico. El modelo consiste en un conjunto de símbolos, llamados bloques. Cada bloque puede tener cero o más puertos de entrada, puertos de salida, y estados. Cada bloque representa un sistema dinámico cuyas entradas, estados y salidas pueden cambiar continuamente y/o discretamente en puntos específicos en el tiempo. Las líneas se utilizan para conectar los puertos de los bloques entre sí, y representan las dependencias de datos entre bloques. Las señales pueden representarse como valores que se desplazan a lo largo de las líneas que conectan los bloques en un diagrama de bloques.

Si todas las entradas, las salidas y los estados del bloque cambian o continuamente o en una velocidad periódica fija, el bloque se considera que es un bloque 'de velocidad única'. Si las entradas, los estados, y las salidas se actualizan, o juntos o por separado en puntos en el tiempo definidos por dos o más velocidades, el bloque se considera que es un bloque 'multi-velocidad'. Si un modelo tiene unos bloques multi-velocidad en el mismo, o dos o más bloques de velocidad única ejecutándose a velocidades diferentes, entonces el modelo en sí mismo es multi-velocidad (frente a velocidad única).

Un bloque se denomina como 'atómico' si su definición funcional está fuera del contexto del modelo en el que se coloca. Simulink™ tiene un conjunto de bloques atómicos predefinidos (por ejemplo, suma, producto, ganancia), y el usuario también puede crear sus propios bloques atómicos a través de unas 'funciones-S' escritas por el usuario. Al ser atómicas, las definiciones funcionales de las 'funciones-S' se especifican fuera del contexto del modelo, por ejemplo, usando un código C o un código MATLAB 'm'. Un bloque 'compuesto' es un bloque cuya definición funcional se especifica a través del modelo usando unos conjuntos de bloques atómicos y compuestos. Simulink™ permite al usuario especificar 'subsistemas', bloques compuestos cuya definición consiste en conjuntos interconectados de bloques predefinidos, funciones-S' escritas por el usuario, y otros subsistemas Simulink™. Los subsistemas pueden anidarse jerárquicamente, definiendo una 'jerarquía del modelo'.

Las velocidades de muestreo Simulink™ proporcionan un mecanismo para especificar la velocidad con que se ejecutan los componentes de un modelo. La **figura 1** representa un ejemplo del uso de las velocidades de muestreo en el control de la ejecución de los componentes del modelo. Por ejemplo, el diseñador puede especificar que un bloque 4 de establecimiento se ejecuta a una velocidad continua, y un bloque 10 de controlador se ejecuta en alguna velocidad periódica, discreta. La ejecución de los componentes del modelo se programa por la infraestructura Simulink™ cuando se simula, o por el sistema operativo para una implementación en tiempo real. No existe una relación causal entre las dinámicas del modelo y la programación de estas velocidades; los instantes en los que los componentes se ejecutan están predeterminados.

Simulink™ soporta la propagación de las velocidades de muestreo. Por ejemplo, las velocidades de los bloques de ganancia de 8 y 12 en la **figura 1** pueden haberse dejado sin especificar, o más bien, especificadas como "Heredadas". En este caso, asumiendo que se han especificado las velocidades de los bloques de establecimiento 4 y de controlador 10, los bloques de ganancia de 8 y 12 heredan sus velocidades de los bloques de establecimiento y de controlador.

Simulink™ también proporciona unos mecanismos para especificar las relaciones causales entre las dinámicas del modelo y la ejecución de los componentes del modelo, incluyendo: subsistemas de función de llamada, subsistemas activados, subsistemas de iterador, subsistemas de acción y subsistemas habilitados. La especificación de las relaciones causales permite a los usuarios especificar la ejecución de los componentes del modelo condicionados en los valores presentes y pasados de las señales y en otros datos del modelo.

Sin embargo, el ámbito de la ejecución condicional se restringe en general a un subsistema como los procedimientos convencionales de especificar las relaciones que no permiten que el ámbito de la ejecución condicional se defina como un conjunto arbitrario de bloques (en comparación con el conjunto de bloques que comprenden un subsistema). Esta es una limitación importante, ya que hay veces en las que es deseable ejecutar simultáneamente un conjunto de bloques que no están en un subsistema y/o no son contiguos en el modelo. Por ejemplo, los procedimientos convencionales de ejecución condicional no permiten la ejecución de varios bloques distribuidos a través del modelo en el encendido o en el apagado. Como resultado, la manera en que pueden estar compuestas las relaciones causales está restringida. Los procedimientos convencionales de especificar las relaciones causales

entre las dinámicas de un modelo y la ejecución de los componentes del modelo no permiten que un usuario habilite la mitad de los bloques en un subsistema y active los otros bloques. Del mismo modo, uno puede no activar algunos de los bloques en un subsistema con un activador, y los bloques restantes con un activador diferente.

Un inconveniente adicional de los mecanismos convencionales para especificar las relaciones causales entre las dinámicas de un modelo y la ejecución de los componentes del modelo es que estos mecanismos requieren unas conexiones gráficas a realizarse en el diagrama de bloques para indicar la causalidad. Esto puede resultar en una apariencia de que algunos usuarios sienten que “desordenan” el diagrama. Otra limitación de los mecanismos convencionales para especificar las relaciones causales es que los mecanismos convencionales no se mapean de manera natural en el software avanzado o en las construcciones del sistema operativo, tal como la inicialización, las excepciones, o las tareas. Del mismo modo, ya que los mecanismos convencionales de especificar las relaciones causales carecen de un primer objeto de clase asociado con la relación causal, es difícil configurar y asignar unas características a esa relación. También es difícil para el usuario aprovechar directamente las dinámicas implícitas asociadas con los mecanismos, por ejemplo, los procedimientos de habilitar y deshabilitar asociados con un subsistema habilitado, y para ejecutar condicionalmente los componentes del modelo basados en estas dinámicas implícitas.

El documento US 5.497.500 trata de la manipulación de eventos por medio de un manipulador de eventos dentro de un entorno gráfico. El documento US 5.497.500 no permite ejecutar los componentes basados en tiempo dentro del entorno gráfico.

El entorno de modelado gráfico que tiene una pluralidad de componentes basados en tiempo ejecutables se conoce en la técnica. El artículo “Real-time workshop - for use with Simulink” (http://mntek3.ulb.ac.be/Matlab/pdf_doc/rtw/rtw_ug.pdf) trata, por ejemplo, en la página 373, un entorno de modelado gráfico. En este caso, se asignan diferentes tiempos de muestreo a diferentes bloques que tienen dependencias de datos. A partir de este artículo, es el problema objetivo de la presente invención permitir un modelado y una estimulación más eficiente de los sistemas dinámicos en los entornos gráficos conocidos.

El problema se resuelve por el procedimiento de la reivindicación 1 y el sistema de la reivindicación 12. Otras realizaciones de la invención pueden obtenerse de las reivindicaciones dependientes.

La realización ilustrativa de la presente invención proporciona un mecanismo para especificar y configurar una relación causal entre las dinámicas de un modelo y la ejecución de los componentes del modelo. La ejecución del componente del modelo está ligada a la aparición de los “eventos del modelo”. Los eventos del modelo se definen primero en el entorno de modelado. La aparición de condiciones en el modelo especificadas en la definición del evento hace que el evento se “envíe”. Los componentes del modelo que se han asociado con la aparición del evento “reciben” la notificación del envío del evento y a continuación se ejecutan. Los componentes aislados dentro de un subsistema pueden designarse para ejecutarse tras la aparición de un evento, como componentes no contiguos dentro de un modelo. La asociación entre los eventos del modelo y la ejecución del componente puede especificarse sin dibujar los indicadores gráficos que conectan los componentes en la vista del modelo.

En el entorno de modelado que tiene al menos un modelo con múltiples componentes ejecutables, el procedimiento monitoriza la ejecución del modelo para la aparición de un evento especificado. Tras determinar la aparición del evento especificado, la aparición del evento se envía a un manipulador de eventos. A continuación, se ejecuta un componente en respuesta a la notificación del manipulador de eventos.

En otra realización, en un entorno de modelado que tiene al menos un modelo con múltiples componentes ejecutables, un procedimiento monitoriza la ejecución del modelo para la aparición de un evento especificado. Tras determinar la aparición del evento especificado, se interrumpe la ejecución de otro evento en respuesta a la determinación de la aparición del evento especificado. A continuación, se realiza una operación en el modelo en respuesta a la determinación de la aparición del evento especificado.

En una realización, en un entorno de modelado, un sistema incluye un modelo gráfico con múltiples componentes ejecutables. El sistema también incluye un manipulador de eventos. El manipulador de eventos recibe una notificación del modelo de la aparición de un evento especificado. El sistema incluye además al menos un componente que recibe la notificación desde el manipulador de eventos de la aparición del evento especificado. El componente de recepción se ejecuta en respuesta a la notificación.

Breve descripción de los dibujos

La **figura 1** representa un modelo de diagrama de bloques convencional que utiliza unas velocidades de muestreo;
la **figura 2** representa un modelo que utiliza el procedimiento de envío de la realización ilustrativa de la presente invención;
la **figura 3** representa el envío y la recepción de unos eventos en un entorno Stateflow™ de acuerdo con la realización ilustrativa de la presente invención;
la **figura 4** es un diagrama de temporización de los eventos implícitos que ocurren en un subsistema habilitado;
la **figura 5** representa un modelo que usa los eventos implícitos en un subsistema habilitado;

la **figura 6A** representa un modelo con unas transiciones de eventos que ocurren dentro de la misma tarea y sin unos bloques de transición de eventos;
la **figura 6B** representa un modelo con unas transiciones de eventos que ocurren en diferentes tareas usando unos bloques de transición de eventos;
la **figura 6C** es una línea de tiempo del uso de los bloques de transición de eventos en un modelo;
la **figura 7A** representa un modelo que utiliza la realización ilustrativa de la presente invención para manipular los eventos como excepciones;
la **figura 7B** representa un modelo que utiliza la realización ilustrativa de la presente invención para controlar el orden de ejecución usando un bloque de prioridad de rama;
la **figura 7C** resalta el orden de ejecución dictado por el bloque de prioridad de rama en el modelo de la **figura 7B**;
la **figura 7D** representa un modelo que utiliza la realización ilustrativa de la presente invención para controlar el orden de ejecución con grupos de bloques;
la **figura 8** es un diagrama de flujo de datos que contrasta la manipulación de los eventos normales y excepcionales en la realización ilustrativa de la presente invención;
la **figura 9** representa un entorno adecuado para practicar la realización ilustrativa de la presente invención; y
la **figura 10** es un diagrama de flujo de una vista de alto nivel de la secuencia de etapas seguidas por la realización ilustrativa de la presente invención para asociar las velocidades de muestreo con la aparición del evento.

Descripción detallada

La realización ilustrativa de la presente invención proporciona un mecanismo para ligar la ejecución de los componentes del modelo a la aparición de los eventos del modelo especificados. Las velocidades de muestreo se especifican como eventos que ligan de este modo la ejecución del componente del modelo a las dinámicas del modelo. Los elementos del modelo no contiguos pueden configurarse para ejecutarse condicionalmente basándose en una aparición del evento del modelo. Además, el ámbito de la ejecución del componente no está limitado a los subsistemas en su totalidad como se requiere por algunos sistemas convencionales. La ejecución condicional de los componentes basados en una aparición del evento también puede usarse para manipular las excepciones. Las asociaciones entre los componentes y los eventos del modelo pueden establecerse sin dibujar las conexiones de componente adicionales en la vista del modelo.

En aras de la claridad, la explicación de la realización ilustrativa de la presente invención contenida en el presente documento hace referencia a un entorno de modelado basado en Simulink™ y MATLAB™ (ambas aplicaciones de The MathWorks de Natick, Massachusetts). Sin embargo, debería reconocerse por los expertos en la materia, que la realización ilustrativa de la presente invención puede aplicarse también a otros entornos de modelado y a otros tipos de diagramas además de a los diagramas de bloques tradicionales como Stateflow™ de The MathWorks de Natick, Massachusetts, una aplicación de diagramas de estado, y unos entornos de diagramas de flujos de datos.

Un evento del modelo Simulink™ tal como se usa en la realización ilustrativa de la presente invención (también denominado en lo sucesivo en el presente documento como “evento” o “evento Simulink™”) puede definirse explícitamente en un espacio de trabajo de MATLAB como un objeto. Sus atributos pueden incluir un nombre, un color, una velocidad esperada opcional, una tarea opcional, y una explicación de cómo debería implementarse la función correspondiente al evento (por ejemplo, en línea frente a una firma definida explícitamente).

Los eventos que se han definido pueden “enviarse” por bloques en el modelo, basados en unas condiciones arbitrarias que define el usuario. Los bloques que “reciben” este evento se ejecutan cuando se envía. “Enviar” se refiere al envío de un mensaje a un manipulador de eventos que indica la aparición de un evento específico. A continuación, se informa a los bloques que se han registrado con o de otro modo enganchado en el manipulador de eventos acerca de la aparición del evento cuando se envía el evento.

La **figura 2** representa un ejemplo de un modelo 20 que utiliza el procedimiento de envío. El evento “A” se ha definido y dado el atributo de color “azul”. En el modelo 20, el bloque de “Envío” 22 se ha configurado (como se ve por la ventana de diálogo en esa figura) para enviar el evento “A” condicionalmente cuando la señal 24 en su puerto 26 de entrada es mayor que cero. El cuadro de diálogo solicita un número de parámetros del usuario para el bloque 22 de envío que incluye el número de entradas 36, la condición 38 bajo la que se ejecuta el bloque, y el nombre asignado al evento 40 del modelo definido. La vista del modelo 20 indica que la velocidad de muestreo del bloque “constante” 28 se ha especificado para ser el evento “A”. Esta velocidad de muestreo (es decir: la aparición del evento “A”) se propaga a los bloques de “retardo de unidad” 30 y de “salida1” 32 que se han establecido para heredar sus velocidades de muestreo. Por lo tanto, los tres bloques reciben el evento “A”. Los tres bloques 28, 30 y 32 pueden asumir el “color” del evento “A” y sombrearse de azul en una visualización a un usuario como una forma de expresar la asociación lógica. Por lo tanto, cuando la salida del bloque 21 “onda sinusoidal” es positiva, “A” se envía por el bloque de “envío”, se ejecutan los tres bloques 28, 30 y 32, y se actualiza el valor en el puerto de salida raíz del modelo.

El modelo representado en la **figura 2** muestra la resolución de algunos de los problemas asociados con la ejecución condicional de los subsistemas que se proporcionan por la realización ilustrativa de la presente invención.

Hay un primer objeto “A” de clase asociado con la relación causal, de manera que es posible configurar y asignar unas características a esa relación tal como el color. El ámbito de la ejecución condicional no se limita a un subsistema. Los tres bloques 28, 30 y 32 que se ejecutan condicionalmente no están agrupados dentro de un subsistema padre y pueden elegirse desde áreas no contiguas del modelo 20. Además, el ámbito de la ejecución condicional emplea una propagación de velocidad (solo el bloque “constante” 28 especifica “A” como su velocidad de muestreo). Esto resulta en un procedimiento de especificar las velocidades de muestreo que es más conciso de lo que sería el caso si cada bloque que recepciona una “A” tuviera que tener su velocidad de muestreo especificada explícitamente como “A”. Además, la asociación de bloques con un evento no requiere una conexión gráfica entre el bloque 22 de envío y el bloque 28 constante para realizarse en el diagrama para indicar la causalidad. La relación causal entre la condición especificada por el bloque 22 de “envío” y la ejecución de los tres bloques 28, 30 y 32 es a través de su referencia común para el objeto de evento llamado “A” 23.

Un “ámbito” de un evento es el ámbito del espacio de trabajo dentro del que existe. Si un espacio de trabajo está asociado con un subsistema o modelo, el ámbito del evento es ese subsistema o modelo. Los bloques solo pueden especificar su tiempo de muestreo como un evento cuando ese evento está en el ámbito de ese bloque.

La **figura 3** representa otro ejemplo de un entorno en el que ocurre el envío y la recepción explícita de eventos, visualizando una gráfica 30 Stateflow™ y un modelo 32 de diagrama de bloques de los componentes de la gráfica con más detalle. En este ejemplo, el usuario define los eventos “A” 34 y “B” 36, con los colores de atributos verde y azul, respectivamente. Ambos eventos 34 y 36 tienen sus velocidades de muestreo especificadas como 20 ms. A continuación, la gráfica 30 Stateflow se ejecuta cada 20 ms y se envían estos eventos en una secuencia periódica bien definida. Los dos bloques 40 y 50 de puerto de entrada tienen sus velocidades de muestreo establecidas en los eventos “A” 34 y “B” 36. El resto de los bloques asumen estos eventos como unas velocidades de muestreo heredadas y se sombrean con el color correspondiente al evento. El conjunto de bloques de color verde 40, 42, 44 y 46 reciben el evento “A” y se ejecutan cuando se envía ese evento. Del mismo modo, el conjunto de bloques de color azul 50, 52, 54 y 56 se ejecutan cuando se envía ese evento “B”.

El conjunto de elementos que manipula cada evento se ejecutan en el orden relativo en el que aparecen en la lista de bloques ordenados del modelo. El orden de la lista de bloques ordenados está determinado por las dependencias de datos. En consecuencia, cada 20 ms se ejecuta la gráfica 30 Stateflow, y la cadena de bloques de color verde 40, 42, 44, y 46 se ejecuta, de izquierda a derecha, seguida de la cadena de bloques de color azul 50, 52, 54 y 56, de izquierda a derecha. Además, ya que la velocidad de muestreo opcional de los eventos se ha especificado explícitamente para que sea de 20 ms, se realiza una comprobación de tiempo de ejecución para afirmar que estos eventos se envían cada 20 ms. Una de las ventajas de la especificación explícita de la velocidad de un evento es que cualquier código generado para esa velocidad puede usar el tiempo de muestreo constante correspondiente en el código generado siempre que se requiera un tiempo transcurrido entre la ejecución sucesiva, en lugar de requerir el uso de temporizadores, lo que normalmente sería el caso.

A diferencia de los eventos explícitos, que se definen como objetos del espacio de trabajo y cuyas condiciones para enviarse se especifican explícitamente por el usuario (por ejemplo, a través del uso de bloques de “envío” Simulink™ o de la lógica Stateflow), los eventos implícitos están implícitos en las construcciones del modelo y, se envían automáticamente en respuesta a la ejecución de dichas construcciones. El usuario no puede enviar eventos implícitos. Sin embargo, el usuario puede manipular los eventos implícitos, lo que significa que el usuario puede definir los componentes del modelo que se ejecutan directamente en respuesta a un evento implícito.

La realización ilustrativa de la presente invención incluye los cinco tipos de eventos implícitos observados en la tabla a continuación:

Nombre del evento	**Tipo de evento**	**Ámbito**	**Localización donde se envía**	**Cuando enviar**
t_s	Iniciar	Todo el modelo	Raíz del modelo	Comienzo de la ejecución del modelo
to	Inicializar	Habilitar/Si/ subsistema de caso	Nivel superior del subsistema	La primera vez el subsistema transiciona de inactivo a activo
t_e	Habilitar	Habilitar/Si/ subsistema de caso	Nivel superior del subsistema	Siempre que el subsistema transiciona de inactivo a activo
t_d	Deshabilitar	Habilitar/Si/ subsistema de caso	Nivel superior del subsistema	Siempre que el subsistema transiciona de activo to inactivo
t_f	Cerrar	Todo el modelo	Raíz del modelo	Fin de la ejecución del modelo

Los expertos en la materia reconocerán que pueden manipularse otros tipos de eventos implícitos, además de los

que enumerados en la tabla anterior sin alejarse del ámbito de la presente invención. Por ejemplo, otros tipos de eventos implícitos que pueden soportarse incluyen condiciones de error, tales como un evento implícito enviado cuando un bloque de producto intenta dividir por cero, o un bloque logaritmo intenta tomar el logaritmo de cero. Los objetos correspondientes a los eventos implícitos pueblan automáticamente los diversos espacios de trabajo del modelo, por lo que sus propiedades pueden configurarse por el usuario.

La **figura 4** representa los eventos implícitos observados en la tabla anterior en un diagrama de temporización. La **figura 4** muestra la temporización relativa de los eventos que están en el ámbito en un subsistema habilitado o sus descendientes en la jerarquía de la función de llamada; la señal variable en el tiempo es la señal de habilitación del subsistema. Los cinco tipos de eventos implícitos tienen una velocidad de muestreo asíncrona. Los eventos implícitos incluyen un inicio 70, una inicialización 72, una habilitación 74, una deshabilitación 76 y un cierre 78.

Un ejemplo del uso de eventos implícitos en un subsistema 90 habilitado se muestra en la **figura 5**. El subsistema 90 tiene unos eventos de habilitar y deshabilitar implícitos asociados, que se envían cuando el subsistema se habilita y deshabilita. El bloque 92 de “retención de orden cero” tiene su velocidad de muestreo especificada como “deshabilitada”, de manera que se ejecuta cuando el evento de deshabilitar implícito se ejecuta, es decir, cuando el subsistema 90 habilitado se deshabilita. A través de la propagación, el bloque “Ganancia1” 94 también hereda ese evento (deshabilitado) como su velocidad de muestreo, y juntos, estos dos bloques se ejecutan y actualizan la memoria asociada con el puerto 96 de salida del subsistema cuando el subsistema 90 se deshabilita.

Cada evento en la realización ilustrativa de la presente invención se mapea 1-1 a una función (también denominada en el presente documento como el “manipulador de eventos”) que sirve como punto de entrada para el código correspondiente a los bloques cuya velocidad de muestreo se ha especificado o heredado como el evento. Siempre que las condiciones que conducen al envío del evento son verdaderas, el sistema reacciona llamando a la función. La función puede dejarse en línea durante la generación del código. La elección de si se deja o no en línea la función de un evento es una propiedad del objeto de evento correspondiente. Como una implementación por defecto, un usuario puede elegir por dejar en línea las funciones de los eventos implícitos, y por no dejar en línea las funciones de los eventos explícitos.

Una de las propiedades de un evento Simulink™ es su tarea. Por defecto, un evento hereda su tarea del contexto desde el que se envía. Por ejemplo, en la **figura 3** los eventos “A” 34 y “B” 36 se ejecutan en la misma tarea que la gráfica 30 Stateflow que envía esos eventos. El código generado llama a las funciones que corresponden a “A” y “B”, como unas llamadas de función local que se representan en el siguiente pseudocódigo correspondiente al modelo de la **figura 3**.

```
void model_step ()
{
 A();
 B();
 }
 void A()
 {
 u3 = x;
 u2’= 2*u1;
 x = u2;
 }
 void B()
 {
 v3 = y;
 v2 = 3*v1;
 y = v2 ;
 }
```

La tarea de un evento puede especificarse como el nombre de un objeto de tarea del modelo, que corresponde a una tarea en la implementación en tiempo real. Los objetos de tarea Simulink™ corresponden a las tareas generadas del sistema operativo. Las propiedades de un objeto Tareas pueden incluir una velocidad designada (un valor periódico, o asíncrono), una prioridad, un tamaño de pila de tareas, si la tarea es preferente, u otras propiedades. Cuando se envía un evento cuya tarea es diferente de la tarea de la construcción de bloque o de modelo que envía el evento, la función correspondiente al evento se programa en la tarea del evento. Si la tarea se especifica para que sea periódica, la función se ejecuta durante cualquier etapa de tiempo de esa tarea durante la que se envía el evento. Si la tarea se especifica como asíncrona, entonces el envío del evento hace que la tarea se ejecute por el sistema operativo.

La realización ilustrativa de la presente invención evita el envío asíncrono de eventos y una implementación multiproceso, a través del uso de tareas. Las tareas se usan para ayudar a garantizar la integridad de los datos cuando se usan los eventos en una implementación en tiempo real indicando dónde se requiere un bloque de transición de evento en aras de la integridad de los datos. Las tareas también se usan para especificar la prioridad relativa de las tareas asociadas con una transición. La realización ilustrativa de la presente invención usa un bloque de transición de eventos que es una generalización del bloque de transición de velocidad de Simulink™. Las figuras

6A, **6B** y **6C** ilustran algunos de los temas involucrados con las transiciones de eventos y el uso de los bloques de transición de eventos.

La **figura 6A** representa un modelo 100 con transiciones que ocurren en respuesta a la aparición del evento A y del evento B. En el modelo 100, los eventos A y B tienen la misma tarea y el problema de integridad de datos puede resolverse utilizando la memoria persistente para almacenar los datos en los límites entre los dos eventos. Ya que los dos eventos están en la misma tarea, no pueden preferirse entre sí, y no son necesarios mecanismos adicionales (por ejemplo, doble almacenamiento intermedio). En este modelo 100, la memoria persistente se utiliza para las salidas de los dos bloques 102, 104 de función de transferencia de la izquierda y los bloques de transición no son necesarios para transferir los datos entre los bloques 102, 104 y 106 de función de transferencia.

Sin embargo, si se especifican los eventos A y B para ejecutarse en diferentes tareas, son necesarios los bloques de transición de eventos, como se muestra en la **figura 6B**. En el modelo 110 de la **figura 6B**, el evento A tiene una tarea con alta prioridad, mientras que el evento B tiene una tarea de baja prioridad. En este caso, el bloque denominado Transición 114, que se encuentra en el límite entre los bloques de función de transferencia primero 112 y segundo 116, actúa como una retención de orden cero. El bloque denominado Transición1 118, que se encuentra en el límite entre los bloques de función de transferencia segundo 116 y tercero 120, actúa como un retraso por defecto.

La línea de tiempo de la **figura 6C** muestra la funcionalidad de los bloques de transición de eventos de la **figura 6B**. Ya que el bloque de transición de eventos Transición 114 actúa como una retención de orden cero, se representa en la figura por unos intervalos de tiempo etiquetados “ZOH”. Ya que el bloque de transición de eventos Transición1 118 actúa como un retraso, se representa en la figura por unos intervalos de tiempo etiquetados “1/z”. El bloque de transición de eventos Transición 114 se ejecuta cada vez que se ejecuta el manipulador para el evento A, siempre que la tarea asignada al evento A no sea preferente a la tarea que contiene el evento B. Esto evita la entrada al manipulador para cambiar B en el medio de su ejecución. Esto es necesario ya que en un sistema asíncrono causal, no hay manera de saber de antemano cuándo el evento B se enviará próximamente. La función de salida para el bloque de transición de eventos Transción1 118 se ejecuta cada vez que el evento A se envía después de que el manipulador para el evento B haya finalizado su ejecución. Esto garantiza que el manipulador para el evento A utiliza el último valor calculado por el manipulador para B, pero no ejecuta la función de salida de Transición1 si no es necesario.

Por lo tanto, en la **figura 6C**, la tarea de baja prioridad asociada con el evento B (etapa 130), ejecuta un retardo (etapa 132), la tarea de alta prioridad asociada con el evento A (etapa 134) se ejecuta y se sigue por la retención de orden cero (etapa 136). A continuación, la tarea asociada con el evento A se ejecuta (etapa 138) sin que se preceda por el retraso ya que B no se ha ejecutado desde la última vez que se ha ejecutado A. A continuación, se ejecuta la retención de orden cero (etapa 140) seguida de la tarea de alta prioridad (etapa 138). A continuación, se ejecuta la tarea de baja prioridad asociada con B (etapa 142) y se interrumpe por la tarea de alta prioridad (etapa 144). Tras la finalización de la tarea de alta prioridad (etapa 144) se reanuda la tarea de baja prioridad (etapa 142) sin que se ejecute la retención de orden cero ya que la tarea de baja prioridad ya había comenzado. El procedimiento continúa con el retardo (etapa 146), la tarea de alta prioridad (etapa 148) y la retención de orden cero (etapa 150) que se ejecutan en orden.

Además de garantizar la integridad de los datos, los bloques de transición de eventos pueden ser necesarios para la resolución durante la propagación de eventos. Sin embargo, al realizar la transición entre los eventos que están en la misma tarea, el bloque copia su valor de entrada a su salida persistente. A continuación, después de la compilación puede intentarse reducir el bloque. Debería observarse que los bloques de transición requieren un parámetro de InitialCondition para inicializar su memoria. En una implementación, el valor por defecto de este parámetro es cero.

Los eventos también pueden usarse para manipular los errores encontrados cuando se ejecuta un modelo. Cuando se produce un error, el modelo puede “enviar excepcionalmente un evento”. Un evento que se envía excepcionalmente se llama un “evento de excepción”. Una característica clave de un evento de excepción es que se manipula de manera diferente a partir de un evento que se envía no excepcionalmente, o un “evento normal”. En particular, cuando un evento envía otro evento de excepción, esa primera función del evento nunca reanuda la ejecución tras salir la función del evento de excepción. En otras palabras, si B interrumpe a A excepcionalmente, la ejecución de A no se reanuda después de la finalización de B. Obsérvese que un evento puede llamarse tanto normal como excepcionalmente en el mismo modelo.

El uso de un evento de excepción en un modelo se representa en la **figura** 7A. En el modelo 160, el componente superior es parte del manipulador para el evento A. Si la magnitud de la entrada u1 162 es menor que o igual que 1e-6, el evento B se envía de manera excepcional, o “se lanza” para abreviar, por el bloque 166 de lanzamiento en el subsistema 164 de excepción. El subsistema 164 de excepción evalúa la entrada 162 y solo lanza el evento B si la entrada u1 162 es menor que o igual que 1e-6. El bloque 166 de lanzamiento es similar en funcionalidad al bloque de envío excepto en que envía un evento de manera excepcional en lugar de normalmente. Cuando se lanza el evento B, se ejecuta el manipulador para el evento B, estableciendo un almacén 168 de datos para una constante fija y grande. Cuando el manipulador para el evento B termina, el control no vuelve al manipulador para el evento A,

ya que el evento B se ha lanzado como una excepción, sino que más bien vuelve al procedimiento de llamada. El código representativo del procedimiento representado en la **figura 7A** puede representarse de la siguiente manera:

```
void model_step ()
{
  A();
  }
  void A()
  {
  u2 = fabs(u1);
  u3 = u2 <= 1.0e-6;
  if (fabs (u1) <= 1.0e-6) {
  B(); /* return from B() exceptionally */
  return;
  }
  v4 = rt_sgn(4*(v1 / u1));
  x = rt_lookup(rtP.lookup, v4);
  }
  void B()
  {
  x = 1.0e6;
  }
```

Si el evento B se hubiese enviado normalmente en lugar de como una excepción, cuando se termina el manipulador para el evento B, el manipulador para el evento A habría finalizado su ejecución. Estos dos escenarios contrastantes se representan de manera abstracta en la **figura 8**. Se entenderá por los expertos en la materia que es importante tener un mecanismo preciso para controlar cómo se clasifican los bloques cuando se introducen eventos en un modelo, ya que los eventos introducen una relación causal antes y después de las relaciones causales implicadas por la dependencia de los datos.

El modelo de la **figura 7A** utiliza un subsistema atómico para forzar una orden de los bloques ordenados en el que el bloque de lanzamiento se ejecuta antes del bloque de producto que está destinado a evitarse condicionalmente a través del evento B de excepción. Debido a que el subsistema atómico es anterior al bloque de producto en la lista de bloques ordenados, sus contenidos, que incluyen el bloque de lanzamiento, se ejecutarán antes que el bloque de producto. Una desventaja de un subsistema atómico es que sus contenidos no se reproducen al mismo nivel gráfico que su gráfico padre, haciendo sus contenidos menos fácilmente discernibles.

Un mecanismo alternativo para el uso de un subsistema atómico para controlar orden de ejecución es asignar la prioridad de las ramas que salen de un bloque, y a continuación todos los bloques en una rama heredar la prioridad de esa rama cuando tal herencia es única y no ambigua. La **figura 7B** muestra la colocación de un “bloque 170 de prioridad de rama” que especifica que los bloques en la rama inferior deberían ejecutarse antes que los bloques en la rama superior. La **figura 7C** indica el orden de ejecución dictado por el bloque 170 de prioridad de rama de la **figura 7B**. El número “1” (172) en el bloque 170 de prioridad de rama indica que las ramas más bajas deberían ejecutarse en primer lugar. El número “2” en el bloque 170 de prioridad de rama indica que la rama superior debería ejecutarse en segundo lugar.

Una alternativa adicional al uso de un subsistema atómico para controlar el orden de ejecución se representa en la **figura 7D**. La **figura 7D** representa el uso de unos “grupos de bloques”. Los bloques en el Grupo 1 (176) aparecerá antes que los bloques en el grupo 2 (178) en la lista de bloques ordenados.

En el diagrama de la izquierda en la **figura 8** que muestra un procedimiento de envío normal para dos eventos, la ejecución de un modelo (etapa 180) activa el evento A (etapa 182). Durante la ejecución del evento A, el evento B ocurre normalmente y se manipula (etapa 184). Después de la manipulación del evento B (etapa 184), se reanuda la manipulación del evento A (etapa 182). Después de la terminación de la ejecución del evento A (etapa 182) el control se devuelve al punto en la ejecución del modelo antes de la aparición del evento A (etapa 180).

En el diagrama de la derecha en la **figura 8**, que muestra un procedimiento de envío de excepción para dos eventos, la ejecución de un modelo (etapa 190) activa el evento A (etapa 192). Durante la ejecución del evento A (etapa 192), se produce el evento B excepcionalmente y se manipula (etapa 194). Debido a que se ha manejado excepcionalmente el evento B, el control pasa de nuevo al punto de la ejecución del modelo que existía antes de la aparición del evento A (etapa 190) sin reanudar la manipulación del evento A.

La “devolución anticipada” asociada con un evento de excepción también puede ayudar a evitar ejecutar inútilmente los cálculos corriente abajo de una señal involucrada en una condición de error. Por ejemplo, en el ejemplo de la **figura 7**, la comprobación de la magnitud de u1 se produce como parte de la ejecución del subsistema atómico antes del bloque de producto y los bloques corriente abajo del bloque de producto. Debido a la devolución anticipada del evento interrumpido, estos bloques se libran de la ejecución (inútil) si se lanza el evento B.

Como ha tratado anteriormente, un evento puede enviarse normal o excepcionalmente. El bloque de envío se usa para enviar un evento normalmente, mientras que el bloque de lanzamiento se usa para enviar un evento

excepcionalmente. Además, puede usarse una API de función-S especificada por el usuario para enviar un evento normal o excepcionalmente. Un componente del modelo puede manipular el evento cuando se lanza normalmente o cuando se lanza excepcionalmente, pero no en ambos casos. Un tiempo de muestreo de un bloque puede especificarse como 'A' para indicar que se manipula el evento A cuando se envía normalmente o 'A.excepción' para indicar que se manipula el evento A cuando se envía excepcionalmente. Un bloque con el tiempo A de muestreo no manipula el evento A cuando se envía excepcionalmente, y un bloque con el tiempo A.excepción de muestreo no manipula el evento A cuando se envía normalmente.

Los eventos explícitos enviados excepcionalmente deberían tener un manipulador no vacío; debería haber al menos un bloque en el modelo que manipulase el evento excepcionalmente. Si este no es el caso, se genera un error. Este requisito se justifica por el reconocimiento de que un evento de excepción explícito se envía por los componentes del modelo que el usuario se ha tomado tiempo para crear, y que un evento de este tipo debería manipularse por el modelo. Los eventos implícitos se envían normalmente si hay un manipulador no vacío. Si ningún componente del modelo está manejando el evento implícito enviado normalmente, se envía el evento excepcionalmente en su lugar. Un bloque de lanzamiento puede reenviar un evento que se está manipulando normalmente como un evento de excepción. Este escenario puede utilizarse cuando no está garantizado el éxito de la manipulación del evento normalmente. Después de un primer intento, el evento puede manipularse excepcionalmente.

La **figura 9** representa un entorno adecuado para practicar la realización ilustrativa de la presente invención. Un dispositivo 200 electrónico mantiene un entorno 202 de modelado gráfico y de ejecución tal como las aplicaciones Simulink™ o Stateflow™. El dispositivo 200 electrónico puede ser una estación de trabajo o un servidor, un ordenador portátil, una PDA, un dispositivo conectado a la red, o algún otro dispositivo electrónico digital capaz de soportar el entorno 202 de modelado gráfico y de ejecución. El entorno 202 de modelado gráfico y de ejecución incluye al menos un modelo 204 gráfico y un manipulador 206 de eventos como se ha tratado anteriormente. El dispositivo 200 electrónico se interconecta con un dispositivo 208 de visualización que visualiza una vista 210 del modelo 204 para un usuario 212. El dispositivo 208 de visualización y el usuario 212 pueden estar localizados local o remotamente del dispositivo 200 electrónico. Los expertos en la materia reconocerán que hay muchas diferentes configuraciones posibles de software y hardware dentro del ámbito de la presente invención.

Una secuencia de alto nivel de etapas seguidas por la realización ilustrativa de la presente invención se muestra en la **figura 10**. La secuencia comienza con la ejecución del modelo 204 en el entorno 202 de modelado gráfico y de ejecución (etapa 220). A continuación, se determina la aparición de un evento especificado anteriormente durante la ejecución del modelo (etapa 222). La aparición del evento se envía al manipulador 206 de eventos (etapa 224). El manipulador de eventos notifica a los bloques registrados cuyas velocidades de muestreo están ligadas a la aparición del evento (etapa 226). Tras la notificación, se ejecutan los bloques cuyas velocidades de muestreo están ligadas a la aparición del evento (etapa 228).

REIVINDICACIONES

1. En un entorno de modelado gráfico que tiene al menos un modelo gráfico ejecutable con una pluralidad de componentes basados en tiempo ejecutables, representando los componentes basados en tiempo unos sistemas dinámicos que tienen entradas, estados y salidas que cambian continua o discretamente en puntos específicos en el tiempo, un procedimiento, que comprende las etapas de:

visualizar una vista del modelo (20) gráfico ejecutable, incluyendo dicho modelo (20) gráfico ejecutable un primer conjunto de componentes basados en tiempo ejecutables, comprendiendo dicho primer conjunto de componentes basados en tiempo ejecutables al menos un componente (22) de envío gráfico ejecutable configurable por el usuario, que tiene al menos un puerto (26) de entrada para recibir al menos una señal (24) de entrada, estando dicho al menos un componente (22) de envío gráfico ejecutable configurable por el usuario configurado para enviar un evento cuando se cumple una condición asociada con dicha al menos una señal (24) de entrada de dicho al menos un componente (22) de envío gráfico ejecutable configurable por el usuario;
asociar al menos un componente (28) basado en tiempo de un segundo conjunto de componentes basados en tiempo con dicho evento;
programar la ejecución del primer conjunto del primer conjunto de componentes basados en tiempo ejecutables en una o más velocidades predeterminadas, identificando cuando se cumple dicha condición durante la ejecución de dicho modelo (20) gráfico ejecutable;
enviar, mediante dicho componente (22) de envío gráfico ejecutable configurable por el usuario, una aparición de dicho evento en dicho entorno de modelado gráfico a un manipulador de eventos;
notificar dicha aparición de dicho evento a dicho al menos un componente basado en tiempo del segundo conjunto que está asociado con dicho evento; y
ejecutar dicho al menos un componente basado en tiempo solo condicionalmente en respuesta a dicha notificación.

2. El procedimiento de la reivindicación 1, que comprende las etapas adicionales de:

registrar el al menos un componente basado en tiempo del segundo conjunto con dicho manipulador de eventos.

3. El procedimiento de la reivindicación 1, que comprende la etapa adicional de:

establecer un tiempo de muestreo para la ejecución inicial de dicho componente basado en tiempo ejecutable del segundo conjunto para ser la aparición del evento especificado.

4. El procedimiento de la reivindicación 3, que comprende la etapa adicional de:

propagar el tiempo de muestreo a un componente basado en tiempo ejecutable adicional de dicho segundo conjunto, estando dicho componente basado en tiempo ejecutable adicional configurado para heredar una velocidad de muestreo.

5. El procedimiento de la reivindicación 3, que comprende la etapa adicional de:

establecer un tiempo de muestreo de una pluralidad de componentes basados en tiempo ejecutables no contiguos del segundo conjunto para ser la aparición de dicho evento.

6. El procedimiento de la reivindicación 5,
en el que dicho tiempo de muestreo para la pluralidad de componentes basados en tiempo ejecutables no contiguos se establece sin ajustar conexiones visibles entre dicha pluralidad de componentes basados en tiempo ejecutables visualizados en dicha vista.

7. El procedimiento de la reivindicación 3, que comprende la etapa adicional de:

indicar con un identificador (ID) de evento en dicha vista que el tiempo de muestreo de dicho componente basado en tiempo ejecutable está establecido para dicho evento.

8. El procedimiento de la reivindicación 1, en el que un ámbito de ejecución del evento para el que se está supervisando la ejecución del modelo gráfico ejecutable está restringido a una parte del modelo gráfico ejecutable.

9. El procedimiento de la reivindicación 1, en el que cada evento en dicho modelo gráfico ejecutable se mapea sobre una base de uno a uno a un manipulador de eventos, siendo dicho manipulador de eventos una función.

10. El procedimiento de la reivindicación 1, en el que un bloque de prioridad de ramas indica una orden de ejecución entre al menos dos ramas de bloques en respuesta a dicha notificación.

11. El procedimiento de la reivindicación 1, en el que se ejecuta más de un segundo conjunto en respuesta a dicha notificación, siendo dichos segundos conjuntos una agrupación de bloques seleccionada por un usuario, especificándose el orden de ejecución de los segundos conjuntos por un usuario.

12. En un entorno de modelado, un sistema, que comprende:

al menos un modelo gráfico con una pluralidad de componentes ejecutables;
un manipulador de eventos, recibiendo dicho manipulador de eventos una notificación de dicho modelo de la aparición de un evento especificado; y
al menos un componente de recepción, recibiendo dicho componente de recepción la notificación desde dicho manipulador de eventos con respecto a la aparición de dicho evento especificado y ejecutándose en respuesta a dicha notificación,
implementando el sistema el procedimiento de acuerdo con una de las reivindicaciones 1 a 11.

Figura 1
(Técnica anterior)

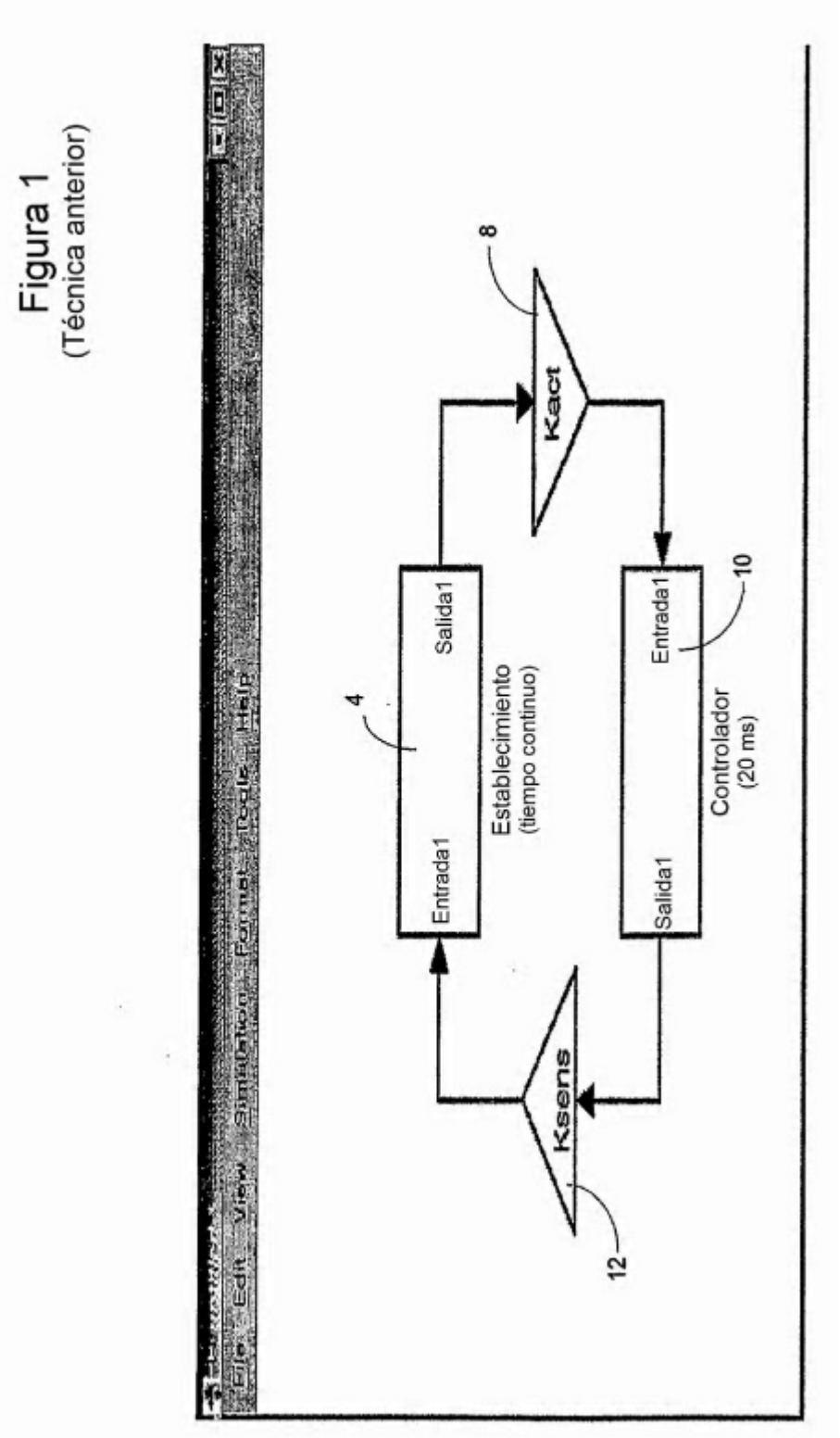


Figura 2

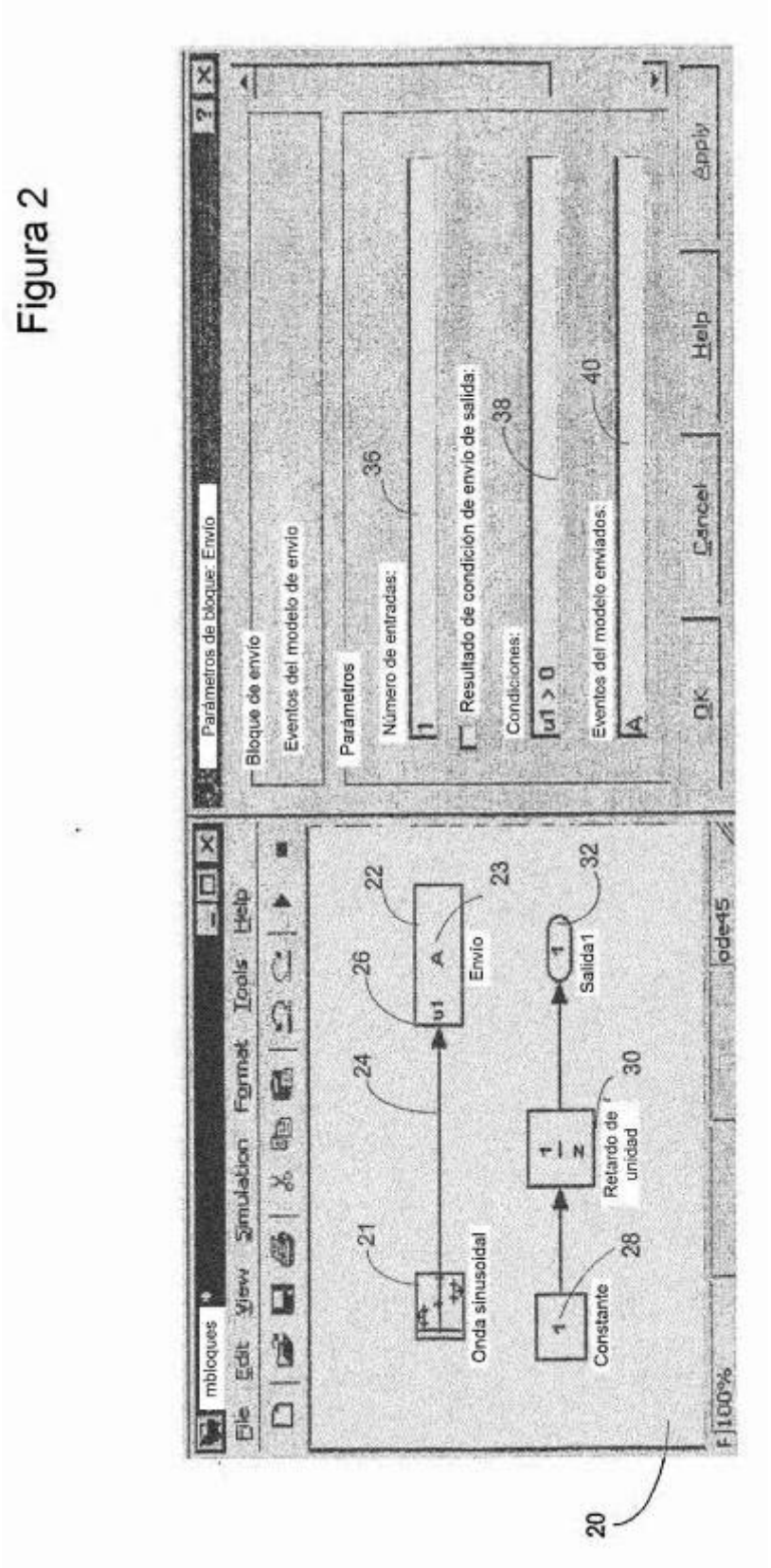

mbloques
File Edit View Simulation Format Tools Help
Onda sinusoidal
Constante
Retardo de unidad
Envío
Salida1
ode45
100%
Parámetros de bloque: Envío
Bloque de envío
Eventos del modelo de envío
Parámetros
Número de entradas:
1
Resultado de condición de envío de salida:
Condiciones:
u1 > 0
Eventos del modelo enviados:
A
OK
Cancel
Help
Apply
20
21
22
23
24
26
28
30
32
36
38
40

Figura 3

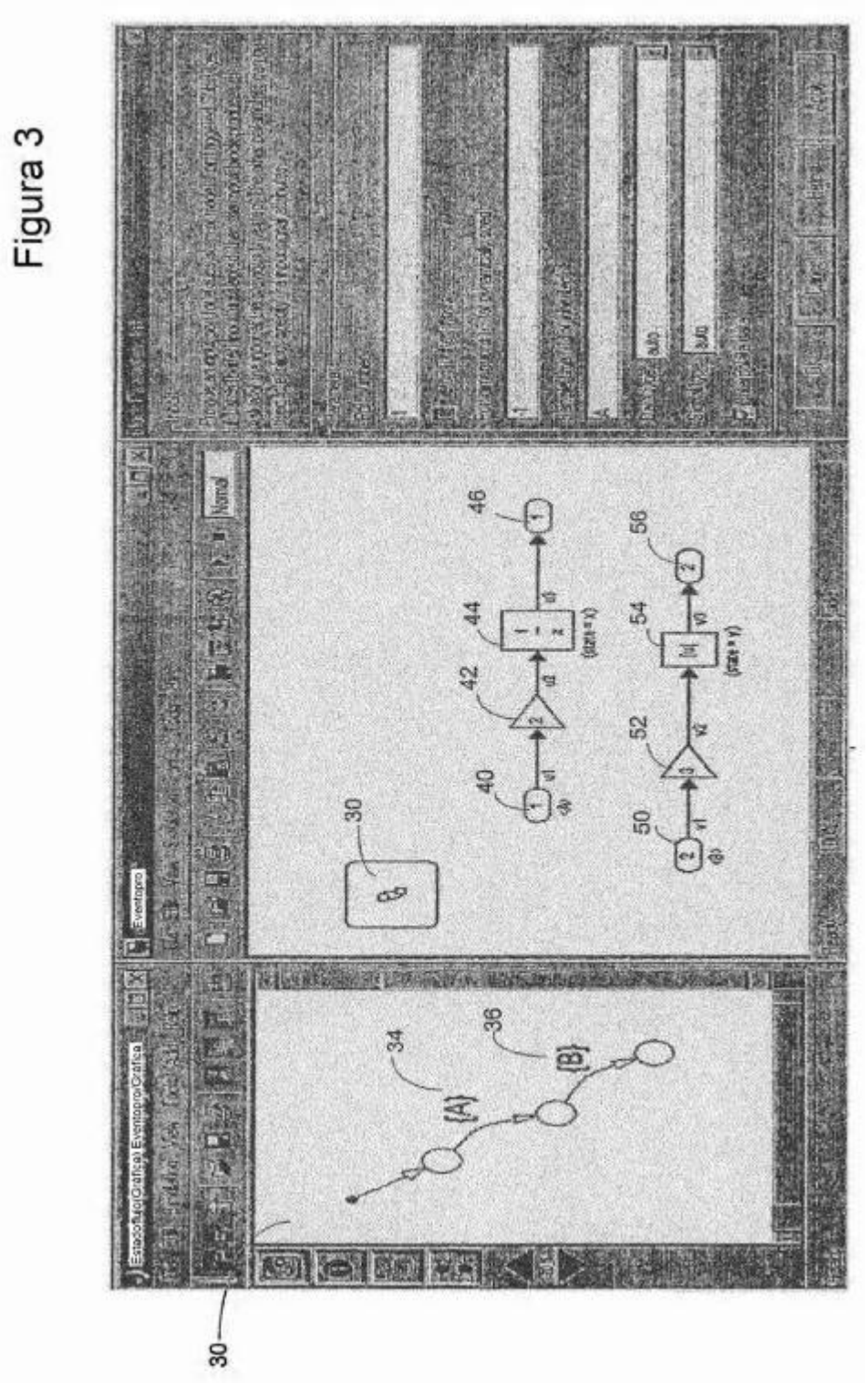


Figura 4

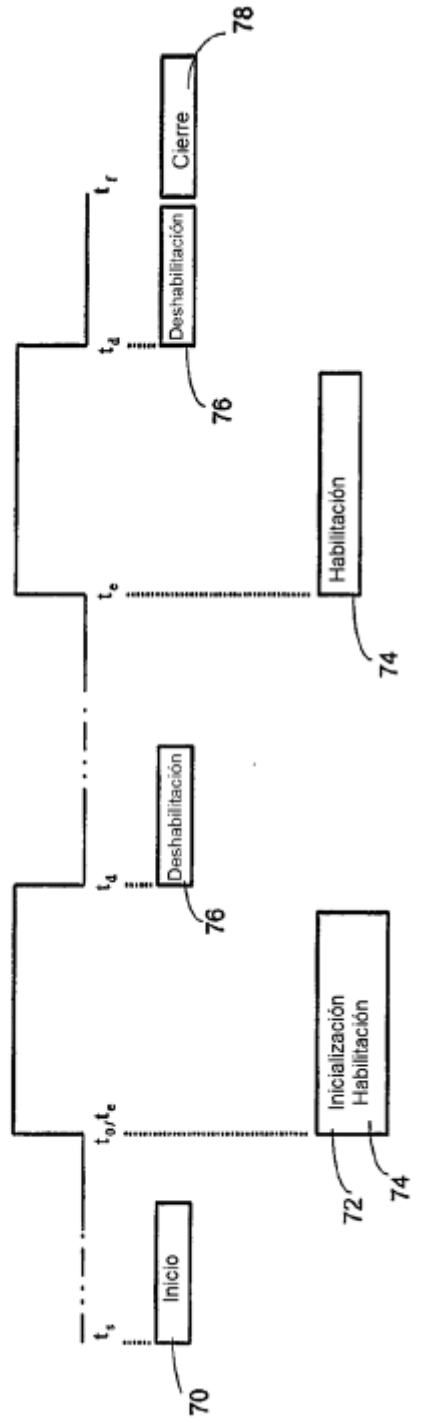

Inicio
70
t_s
Inicialización
Habilitación
72
74
t_0/t_e
Deshabilitación
76
t_d
Habilitación
74
t_e
Deshabilitación
76
t_d
Cierre
78
t_f

Figura 5

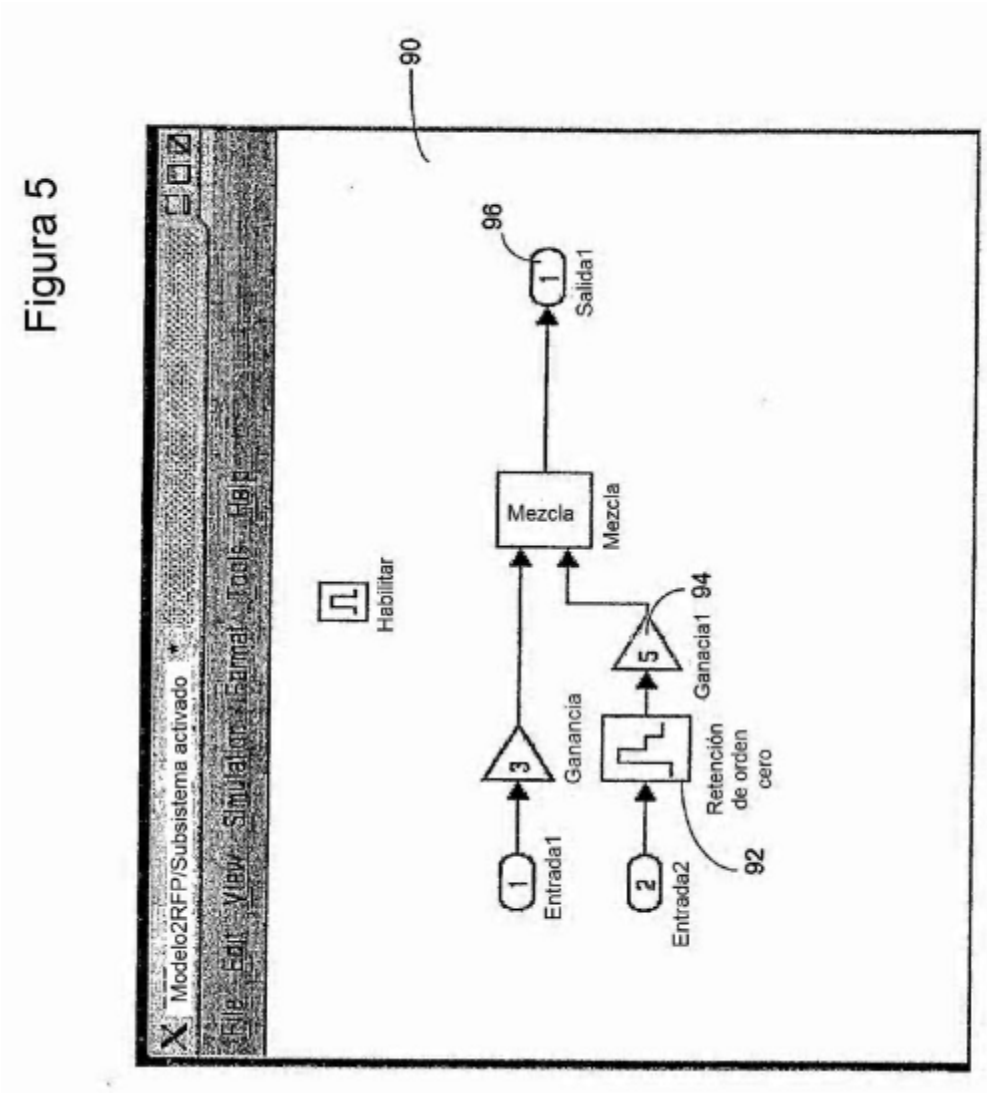

90
Modelo2RFP/Subsistema activado
File Edit View Simulation Format Tools Help
Habilitar
96
1
Salida1
Mezcla
Mezcla
3
Ganancia
5
Ganacia1
94
Retención de orden cero
92
1
Entrada1
2
Entrada2

Figura 6A

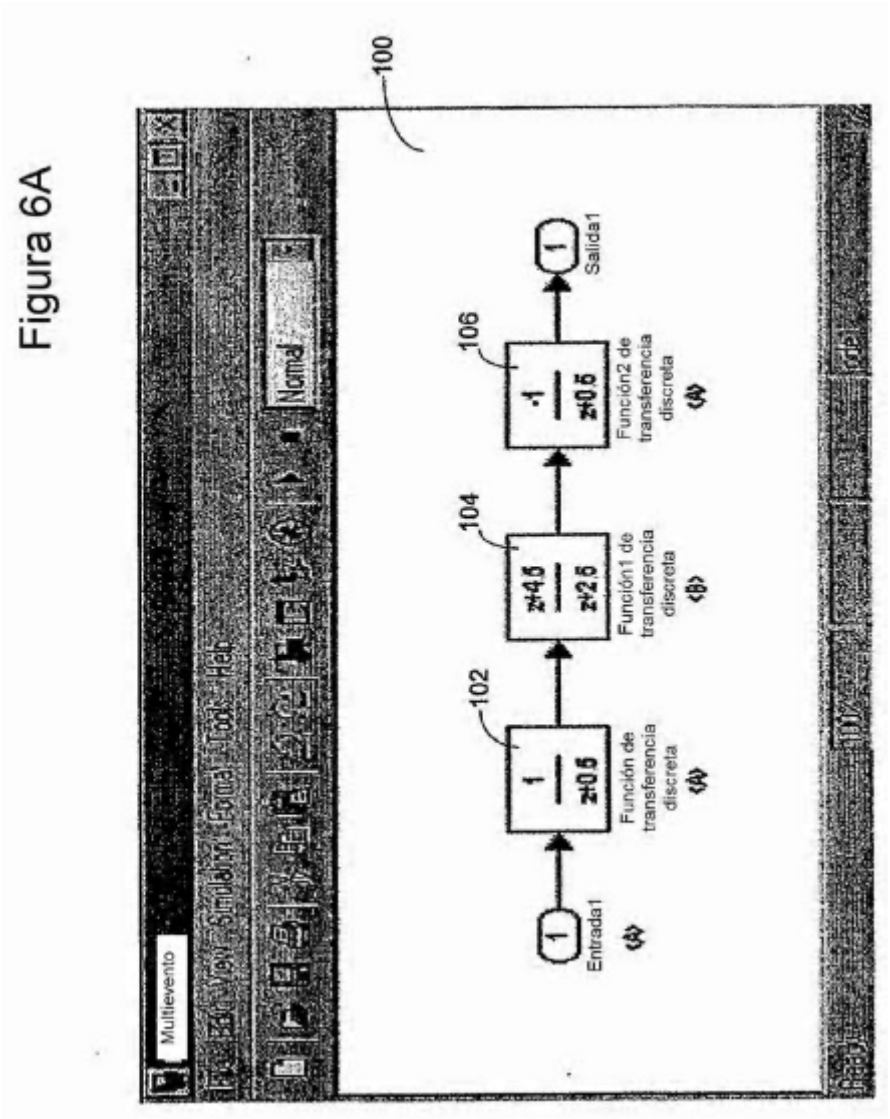

100
Multievento
Normal
102
104
106
Entrada1
Función de transferencia discreta
Función1 de transferencia discreta
Función2 de transferencia discreta
Salida1

Figura 6B

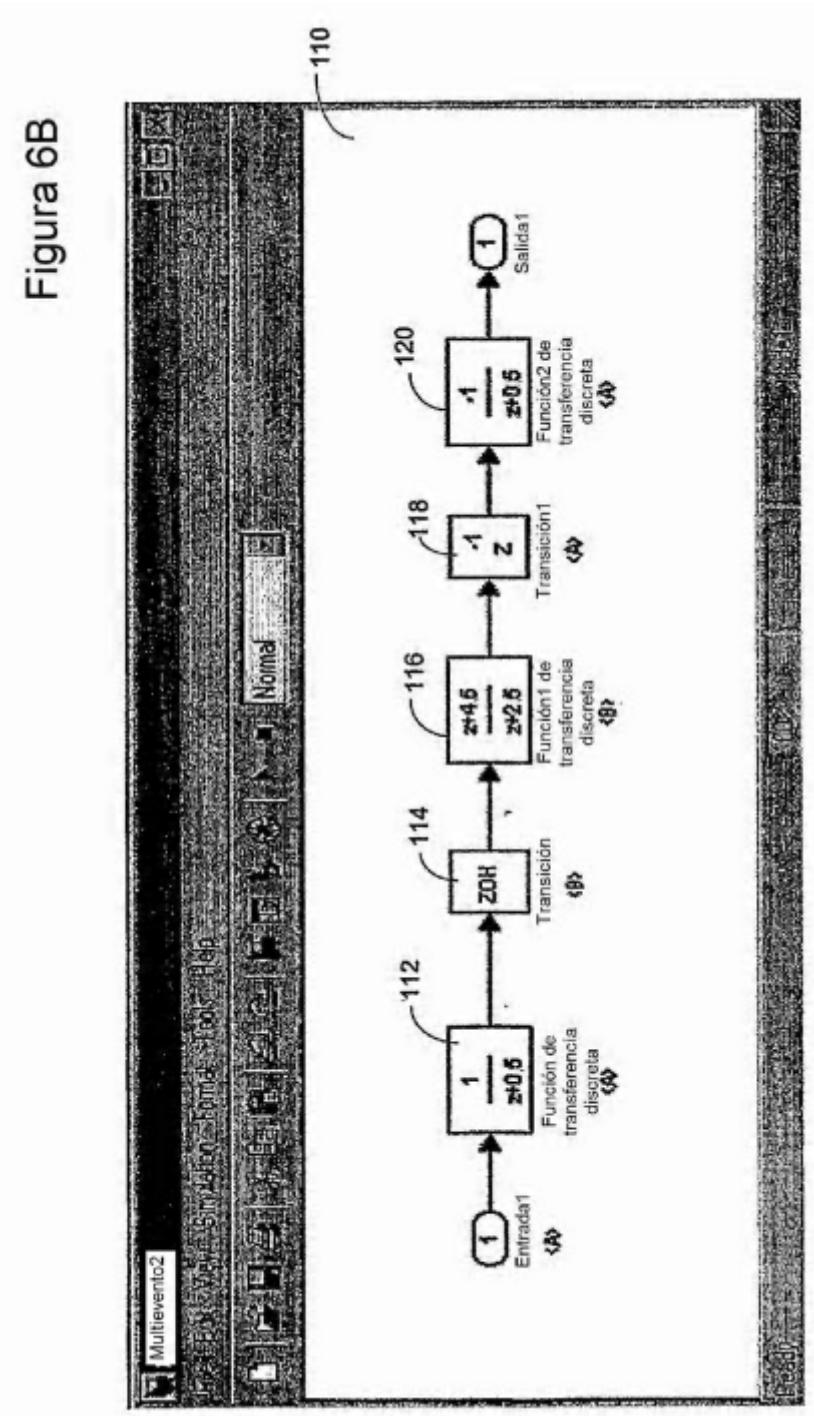

110
Multievento2
Normal
112
1
z+0.5
Función de transferencia discreta
Entrada1
114
ZOH
Transición
116
z+4.5
z+2.5
Función1 de transferencia discreta
118
z
Transición1
120
z+0.5
Función2 de transferencia discreta
Salida1

Figura 6C

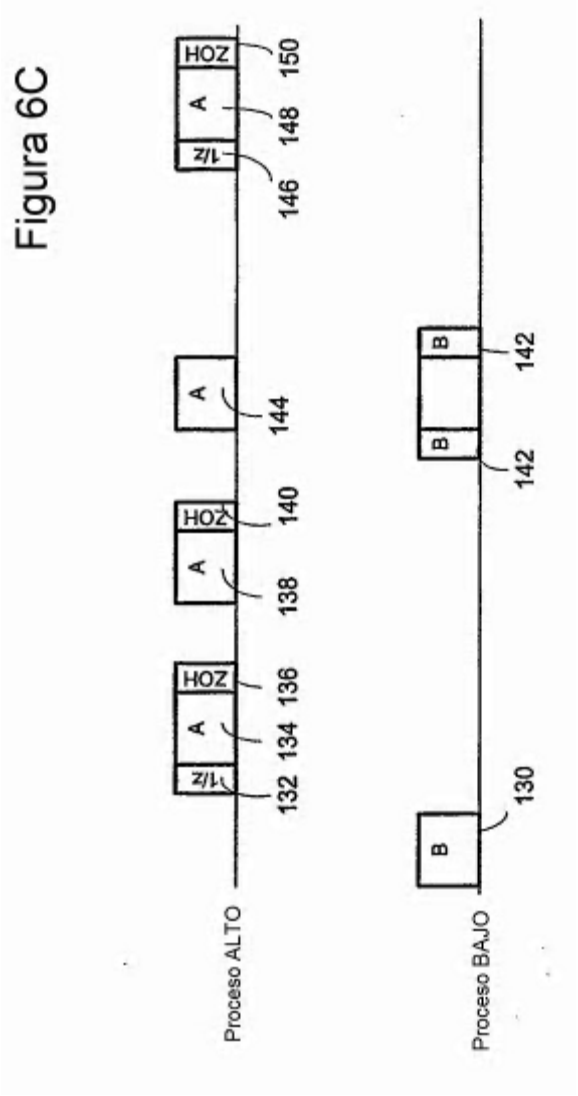

Proceso ALTO
Proceso BAJO
1/2
A
HOZ
B
132
134
136
138
140
142
144
146
148
150
130

Figura 7A

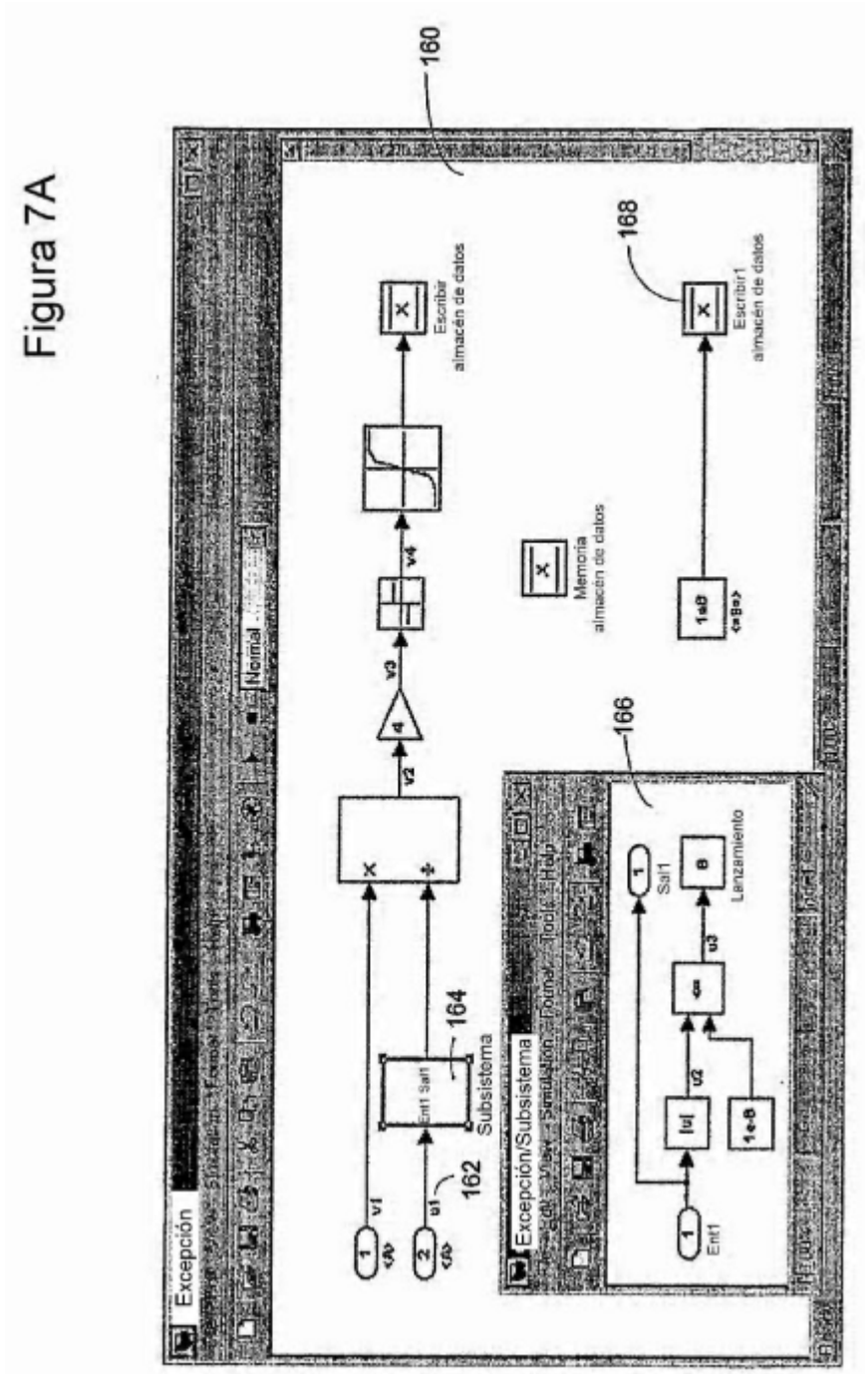

160
Excepción
Normal
Escribir
almacén de datos
168
Escribir1
almacén de datos
Memoria
almacén de datos
166
Excepción/Subsistema
Sal1
Lanzamiento
Ent1
164
Ent1 Sal1
Subsistema
162

Figura 7B

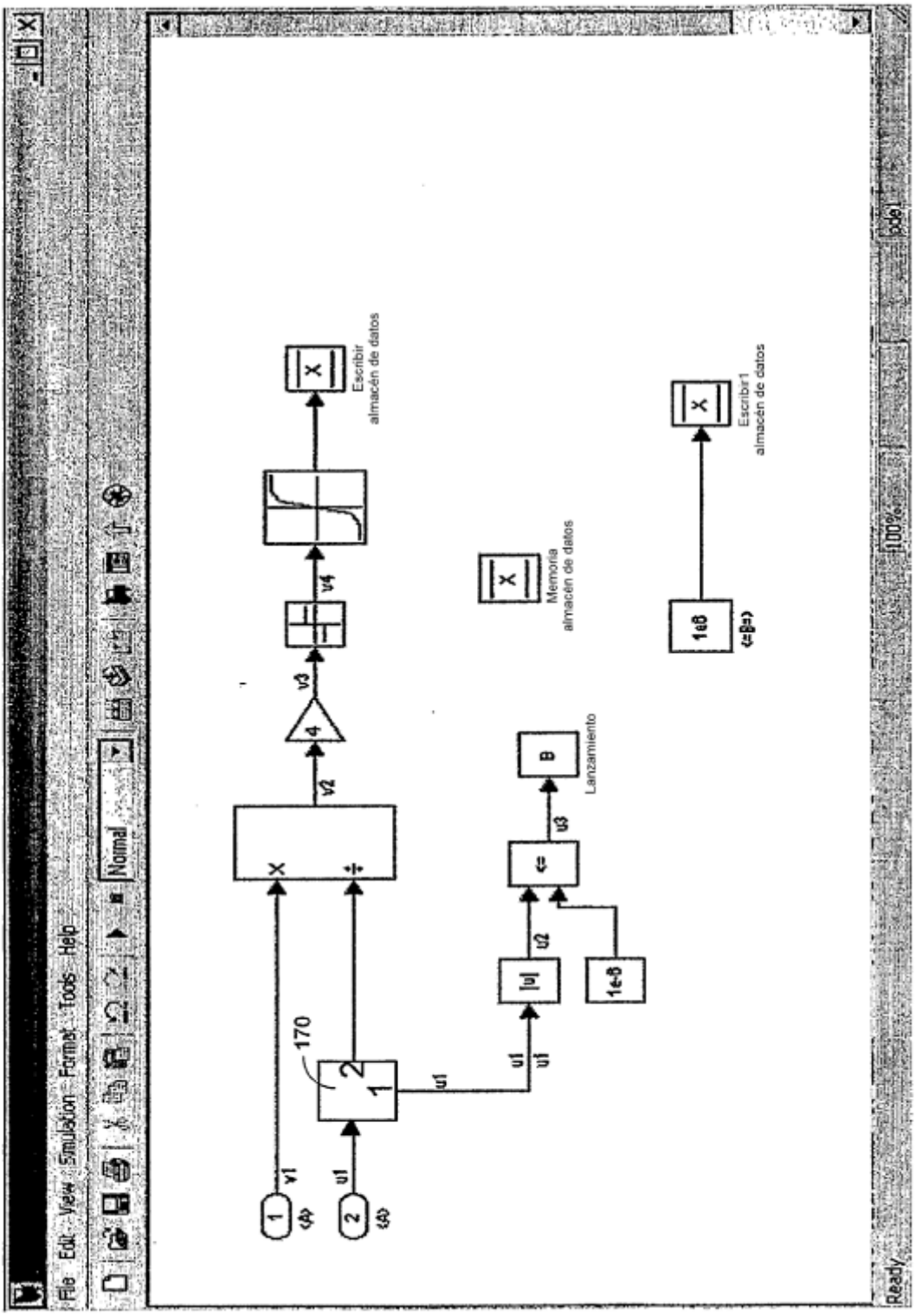

File Edit View Simulation Format Tools Help
Normal
Ready
100%
Escribir
almacén de datos
Memoria
almacén de datos
Escribir1
almacén de datos
Lanzamiento
170
1e-8
1e8

Figura 7C

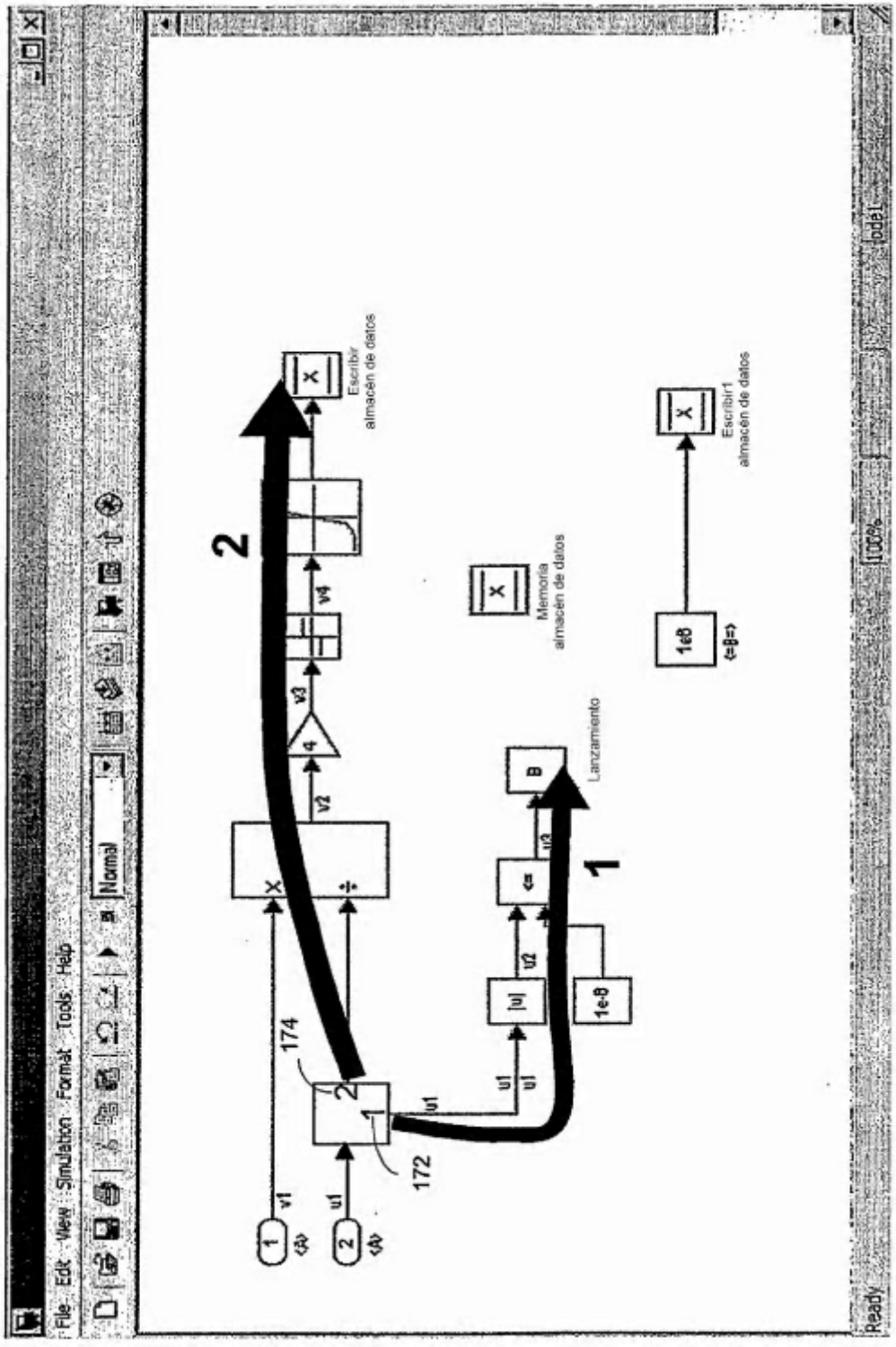

File Edit View Simulation Format Tools Help
Normal
Ready
100%
ode1
Escribir
almacén de datos
Memoria
almacén de datos
Lanzamiento
Escribir1
almacén de datos
174
172
1e-8
1e8

Figura 7D

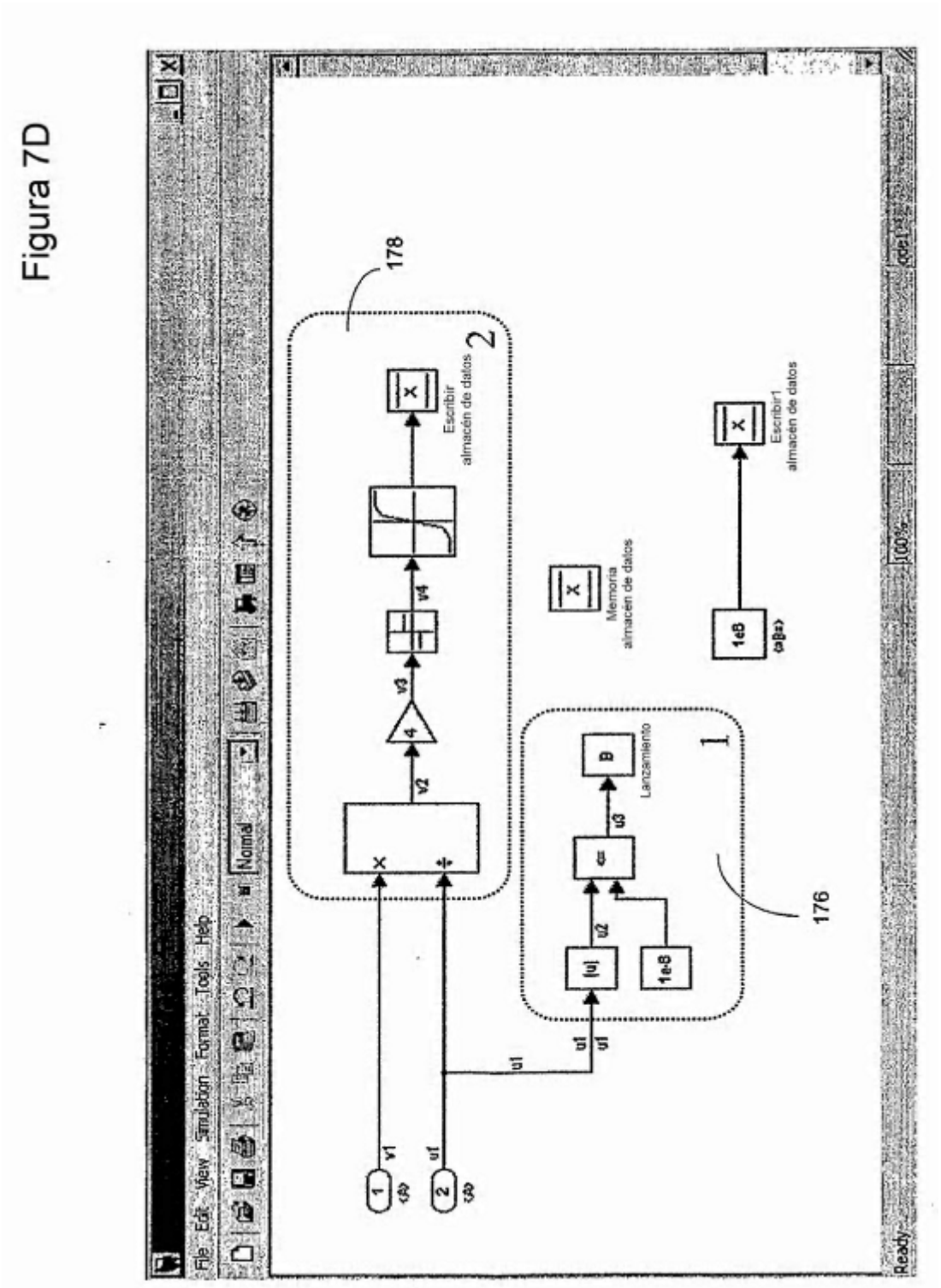
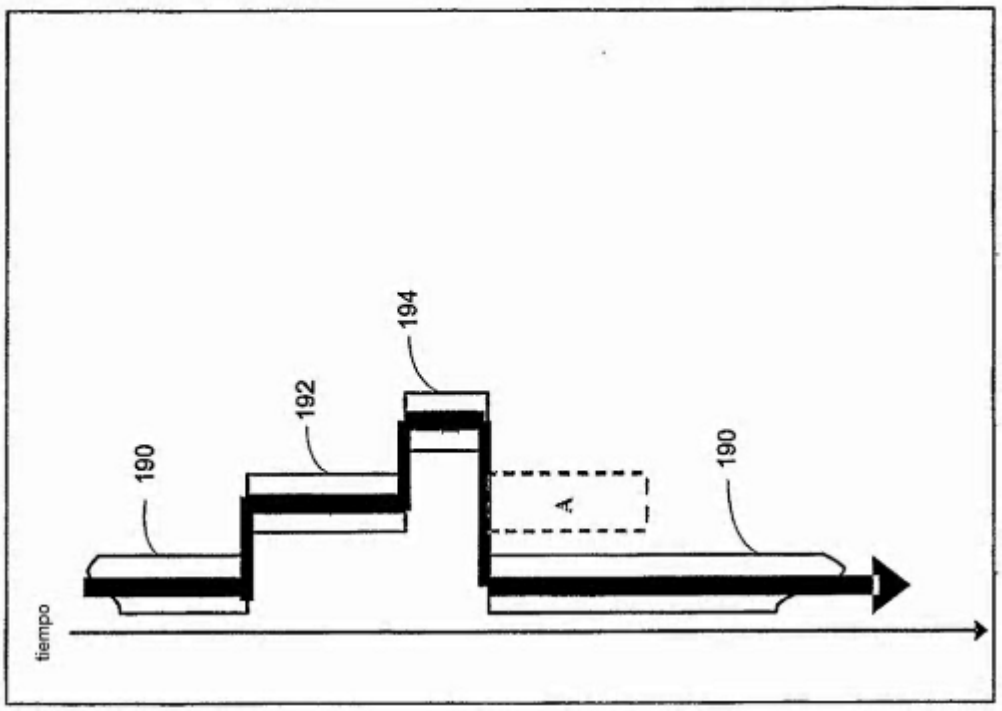

178
176
Escribir
almacén de datos
Memoria
almacén de datos
Escribir1
almacén de datos
Lanzamiento
Normal
File Edit View Simulation Format Tools Help
Ready

Figura 8



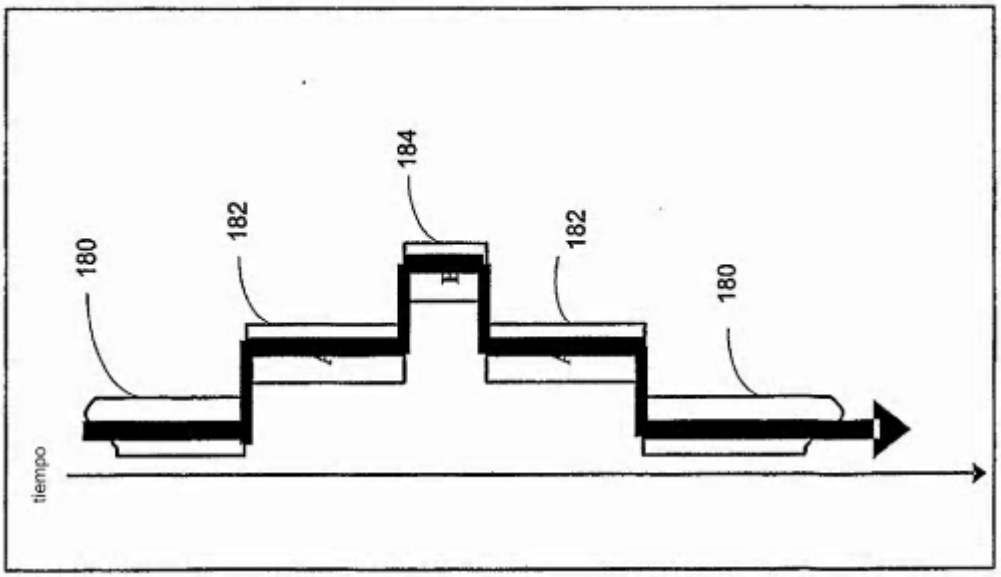


Figura 9

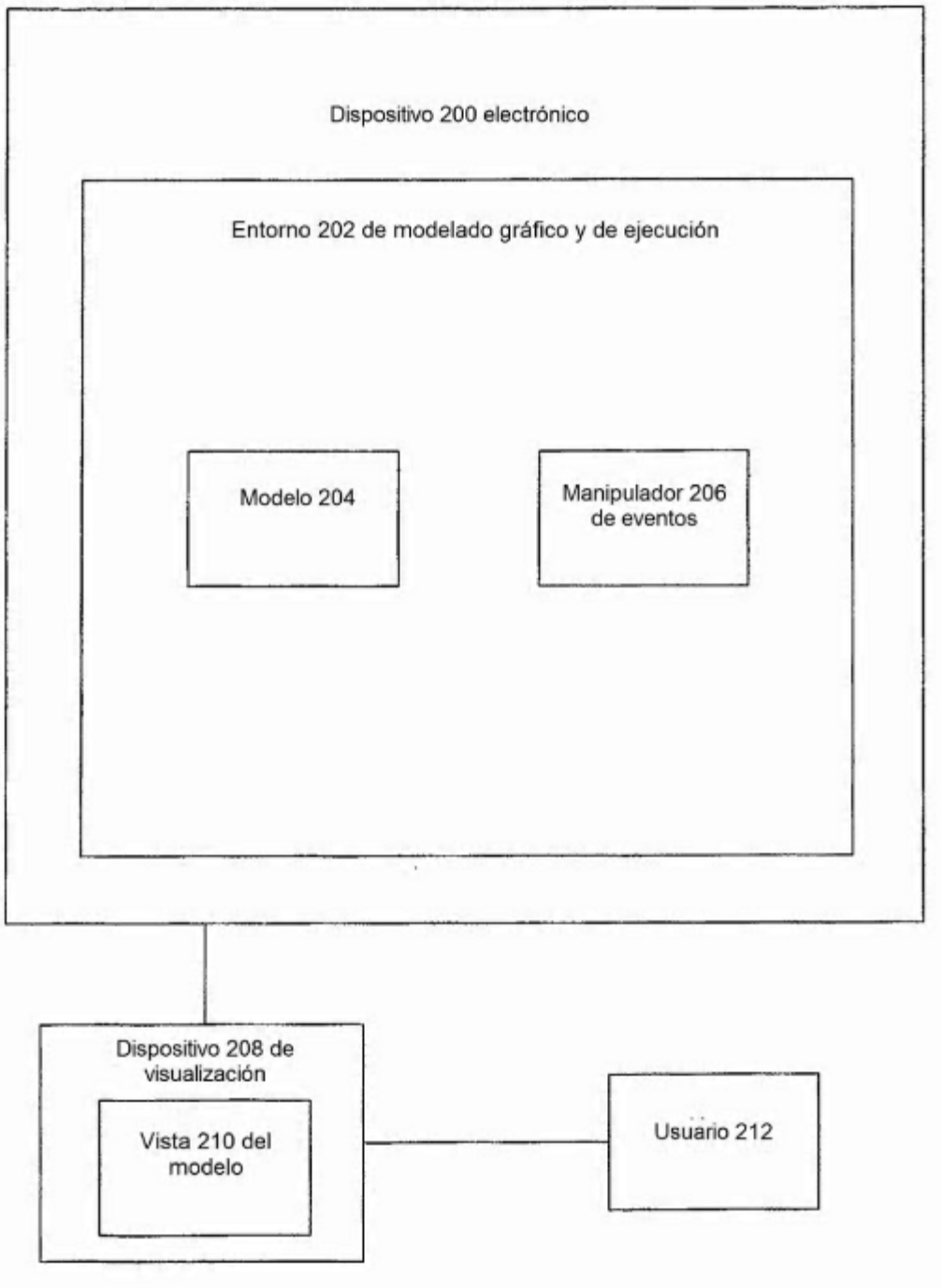
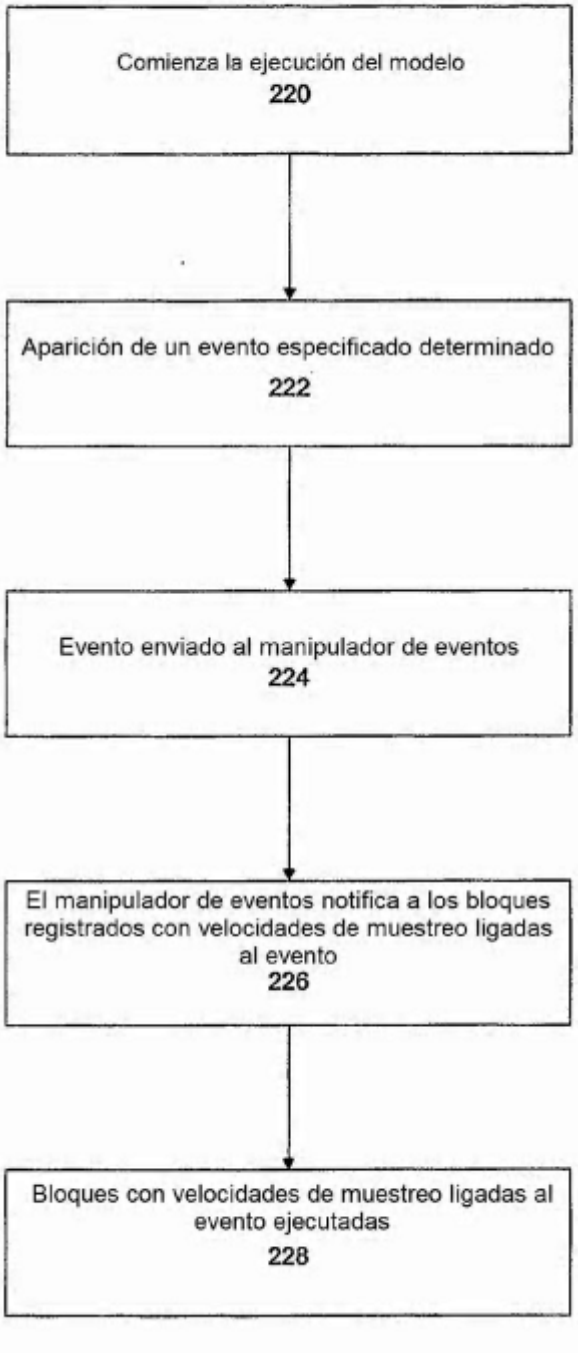

Dispositivo 200 electrónico
Entorno 202 de modelado gráfico y de ejecución
Modelo 204
Manipulador 206 de eventos
Dispositivo 208 de visualización
Vista 210 del modelo
Usuario 212

Figura 10


Comienza la ejecución del modelo
220
Aparición de un evento especificado determinado
222
Evento enviado al manipulador de eventos
224
El manipulador de eventos notifica a los bloques registrados con velocidades de muestreo ligadas al evento
226
Bloques con velocidades de muestreo ligadas al evento ejecutadas
228