Fiuba
Facultad de Ingeniería
Universidad de Buenos Aires
Departamento de Computación
Técnicas de Diseño – 75.10

Trabajo Práctico Final

Entrega Final

17 de junio de 2010

1º Cuatrimestre 2010

Grupo Nº: 3

Integrantes:

78938	García Jaime, Diego
88817	Invernizzi, Esteban Ignacio
88435	Meller, Gustavo Ariel
84623	Risaro, Santiago

Contenido

Enfoque

El objetivo del trabajo práctico fue volcar los contenidos aprendidos tanto en las clases teóricas como prácticas en el diseño y la programación de un juego.

El enfoque principal del juego está en la simpleza de uso y el entretenimiento, desde este punto de vista se intentaron simplificar las interfaces lo más posible, para que no sea complicado utilizar la aplicación.

Se busco lograr una interfaz amigable con el usuario, la cual muestre los resultados de las operaciones en forma práctica y permita un manejo entendible y sin complicaciones.

Se trataron de resolver los problemas más complicados con soluciones analizadas y pensadas por los integrantes, planteando y debatiendo las diferencias y los supuestos de cada uno, logrando decisiones consensuadas y con una mayor riqueza.

En cuanto al entretenimiento se buscó maximizar las posibilidades de utilización de la herramienta mediante la inclusión de múltiples laboratorios y máquinas con varias entradas, de esta manera el usuario cuenta con más posibilidades a la hora de crear una fábrica, pudiéndola adaptar a sus gustos.

Siempre se pensó en que lo que se estaba programando es un juego, motivo por el cual se trató de agregarle variantes y propuestas para enriquecerlo y hacerlo más divertido para el usuario.

De esta forma, se pudo llegar a un acabado sobrio. El sistema resulta intuitivo y uno puede familiarizarse con facilidad. Gracias a esto, las ganas del usuario estarán centradas en el objetivo del juego.

Descripción de la Arquitectura en General

Se ha optado por una arquitectura lo más sencilla posible, siguiendo los lineamientos del patrón MVC. Como es común en aplicaciones de escritorio, la vista y el control están acoplados pues los elementos que muestran información a su vez sirven para manipular el modelo.

Se decidió utilizar la API SWT (http://www.eclipse.org/swt/) para realizar la interfaz de usuario ya que es más simple, directa y presenta un mejor resultado visual y mejor performance que Swing o AWT.

Para simplificar el proceso de compilación y configuración se utilizó maven (http://maven.apache.org/), herramienta que ejecuta los tests con cada compilación y se encarga del manejo de dependencias.

Dentro de la Arquitectura General se trabajo con una sección denominada main, en donde se implementaron todos los requerimientos del trabajo y una sección tests donde se implementaron todas las pruebas unitarias.

Dentro de la sección denominada main, decidimos dividir la aplicación en siete paquetes principales con el objetivo de mantener las incumbencias lo más separadas posibles, los paquetes en cuestión son:

- calendario
- interfazgrafica
- laboratorio
- lineaproduccion
- productos
- jugador
- util

Cada paquete contiene las clases principales destinadas a resolver los requerimientos planteados por el grupo los cuales fueron divididos en secciones.

El paquete calendario contiene todas las clases utilizadas para simular el paso del tiempo y la notificación del mismo al resto del modelo.

El paquete interfazgrafica contiene todas las clases utilizadas para la realización de la interfaz gráfica del juego.

El paquete laboratorio contiene todas las clases utilizadas para la implementación del laboratorio, junto a los diferentes procesos que contiene el mismo.

El paquete lineaproduccion contiene todas las clases utilizadas para la implemetación de la linea de producción: tipos de máquina, maquinas, materias primas, fuente, sumidero, cintas.

El paquete productos contiene todas las clases utilizadas para la implementación de las materias primas y los productos que existirán.

El paquete jugador contiene todas las clases utilizadas para la implementación de un jugador, junto a la fábrica que puede obtener el mismo.

El paquete util contiene las clases utilizadas para la implementación del parser XML y otros recursos de la aplicación.

Decidimos uniformizar el código utilizando los lineamientos de la Sun Code Convention que nos parecieron más relevantes.

Problemas de Dominio

Laboratorio – Inversión

El problema a solucionar era cómo habilitar nuevos procesos a medida que se realizaba una inversión de dinero en el laboratorio por parte del jugador.

Para la inversión de dinero en el laboratorio hemos tomado como hipótesis principal que cada jugador tiene un laboratorio asociado. El mismo puede ser elegido de una serie de laboratorios estándar. Al elegirlo el jugador pasa a estar asociado a ese laboratorio y cuando el jugador lo desee podrá habilitarlo y al invertir dinero podrá descubrirse nuevos procesos.

Este laboratorio tendrá una lista con los procesos que fueron descubiertos y una lista con los que no fueron descubiertos. Inicialmente contará con algunos procesos en la lista de descubiertos y el laboratorio podrá validar líneas de producción contra estos procesos. Estos procesos iniciales son cargados por un Cargador de Procesos el cual levantará todos los procesos existentes.

Mientras que no se encuentre habilitado no se "descubrirán" procesos, es decir que no se podrán pasar procesos de la lista de no descubiertos a la lista de procesos descubiertos.

El jugador podrá "habilitar" el laboratorio sin costo alguno. A partir de allí, el jugador invertirá un porcentaje fijo de su dinero acumulado y esto se irá acumulando en el dinero del laboratorio. A medida del paso del tiempo el laboratorio verificará si puede "habilitar" un nuevo proceso, es decir sacarlo de la lista de no descubiertos y ponerlo en la lista de descubiertos, siempre y cuando tenga el dinero suficiente para habilitarlo ya que cada proceso tiene un costo (según las máquinas incluidas en el proceso). Si cuenta con el dinero suficiente para habilitar algún proceso, se descontará el dinero que cueste ese proceso del laboratorio y se cambiará de lista ese proceso.

Las clases participantes son las siguientes:

- Jugador: Representa a un jugador, el cual tiene su dinero y toma decisiones a cerca de su fabrica.
- Laboratorio: Representa a un laboratorio, el cual tiene procesos descubiertos y procesos no descubiertos y puede descubrir nuevos procesos.
- Proceso: Representa a un proceso, un conjunto de tipos de maquinas con sus entradas y sus salidas, sabe que producto fabrica.
- CargadorDeProcesos: Carga los procesos según un tipo de laboratorio.

Diagrama de Clase Laboratorio – Inversión

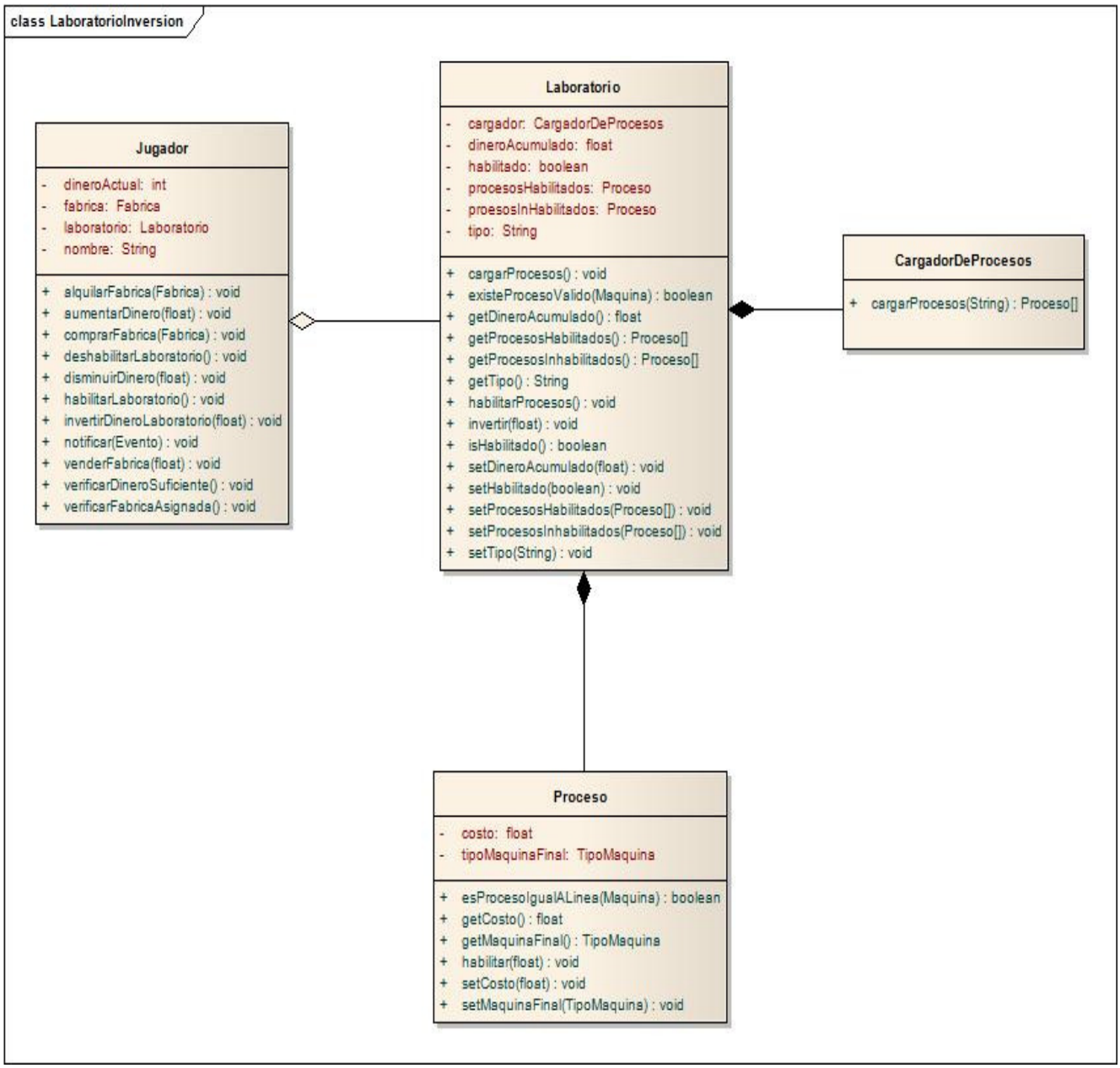
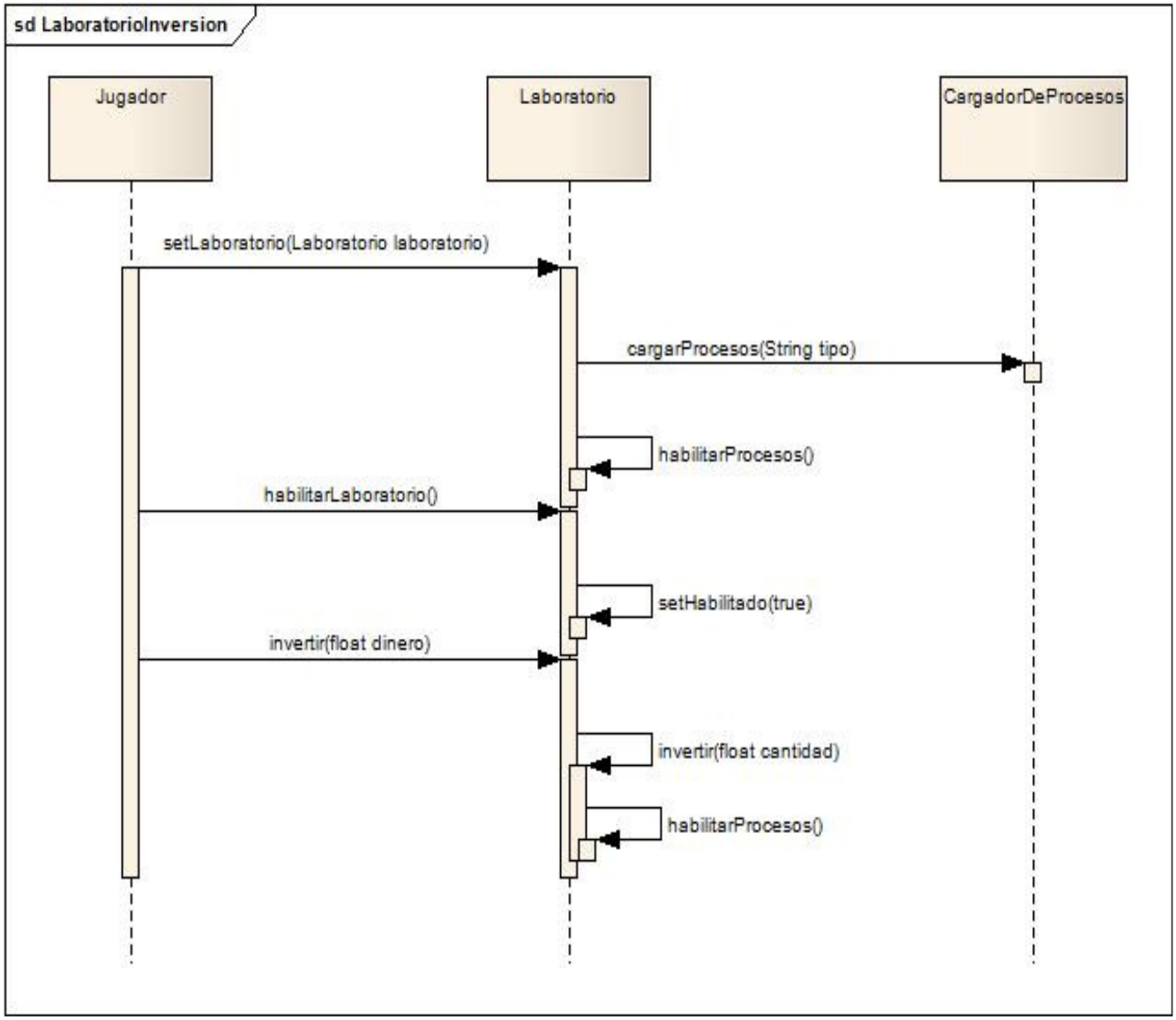


Diagrama de Secuencia Laboratorio – Inversión



Laboratorio – Validación

El problema a solucionar era como habilitar nuevos procesos a medida de una inversión de dinero en el laboratorio por parte del jugador.

Para la validación de procesos el laboratorio contará con una lista de procesos habilitados los cuales dan como resultado un producto específico.

La idea es recibir la máquina final de una línea de producción. Por cada proceso que se encuentre en la lista de descubiertos del laboratorio se comparará cada tipo de máquina con la maquina. Comparando sus maquinas precedentes y materias primas ingresantes.

Lo primero que se ve es si tienen la misma cantidad de materias primas precedentes, es decir si uno tiene más o menos materias primas que el otro no son lo mismo con lo cual esa línea de producción no corresponde a ese proceso.

En caso de tener la misma cantidad de materias primas se compara cada tipo de materia prima para ver si tienen las mismas. Si son las mismas entonces se dará una equivalencia, sino ya se frena el proceso.

Después se ve si tienen la misma cantidad de maquinas precedentes, es decir si uno tiene más maquinas o menos máquinas que el otro no son lo mismo con lo cual esa línea de producción no corresponde a ese proceso.

En caso de tener la misma cantidad de máquinas precedentes, se comparará cada tipo de máquina de ese proceso contra las máquinas de la línea, viendo si el tipo de máquina del proceso y la máquina de la línea de producción son del mismo tipo. Si son del mismo tipo y tienen los mismos tipos de máquinas precedentes entonces habrá una equivalencia. (Todavía no se implemento la comparación de entradas).

Se indicará a la línea de producción que si responde o no a ningún proceso válido.

Las clases participantes son las siguientes:

- Laboratorio: Representa a un laboratorio, el cual tiene procesos descubiertos. Puede comparar sus procesos descubiertos contra una línea de producción y decir si son lo mismo.
- Proceso: Representa a un proceso, un conjunto de tipos de maquinas con sus entradas y sus salidas, sabe que producto fabrica. Puede compararse contra una línea de producción (un conjunto de máquinas) para ver si son lo mismo.
- TipoMaquina: Representa a un tipo de máquina genérico.

- ComparadorDeMaquinas: Interfaz para el manejo de comparaciones de máquinas.
- ComaradorDeMaquinasSimple: Compara una maquina con otra maquina.

Diagrama de Clase Laboratorio – Validación

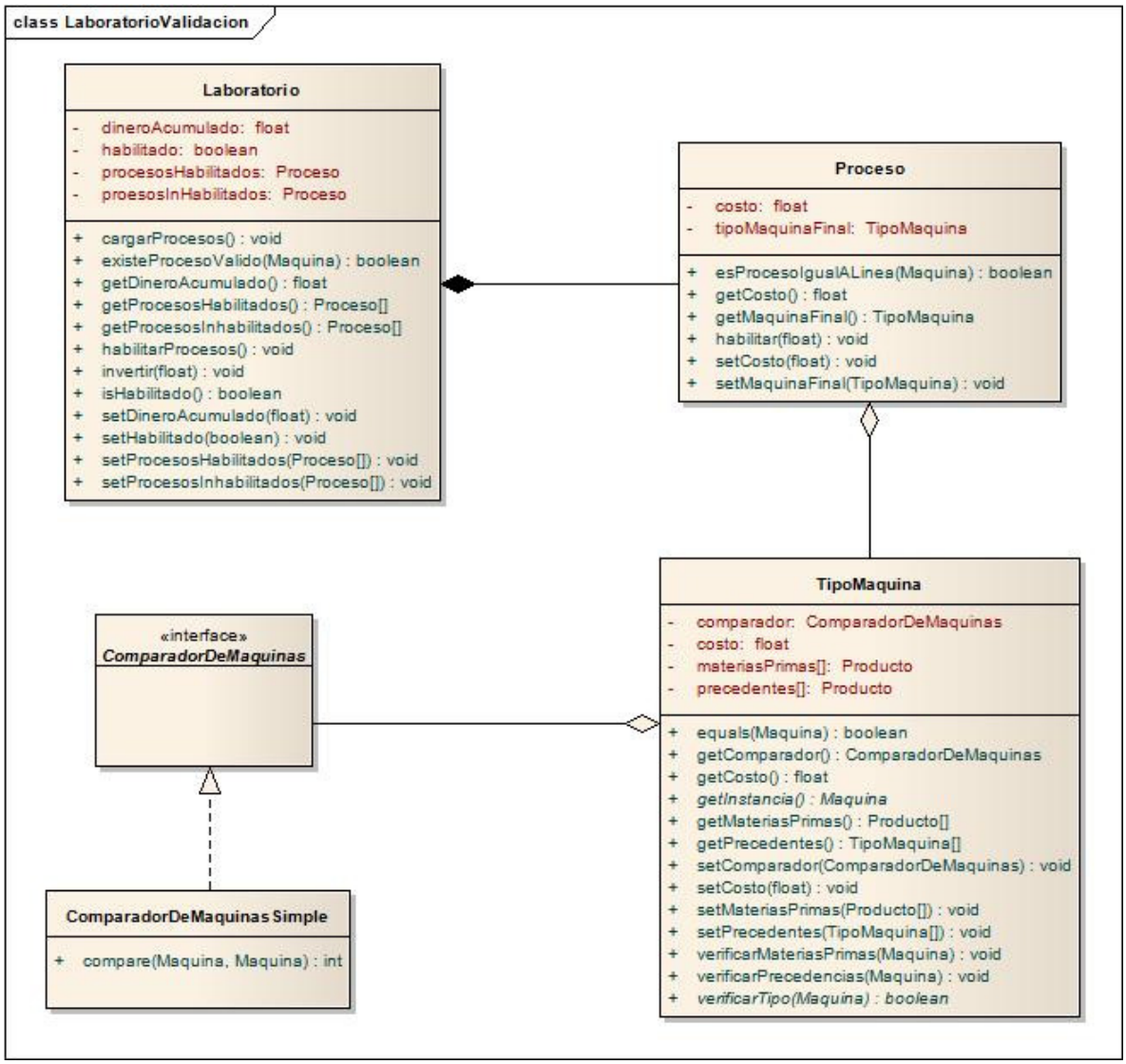
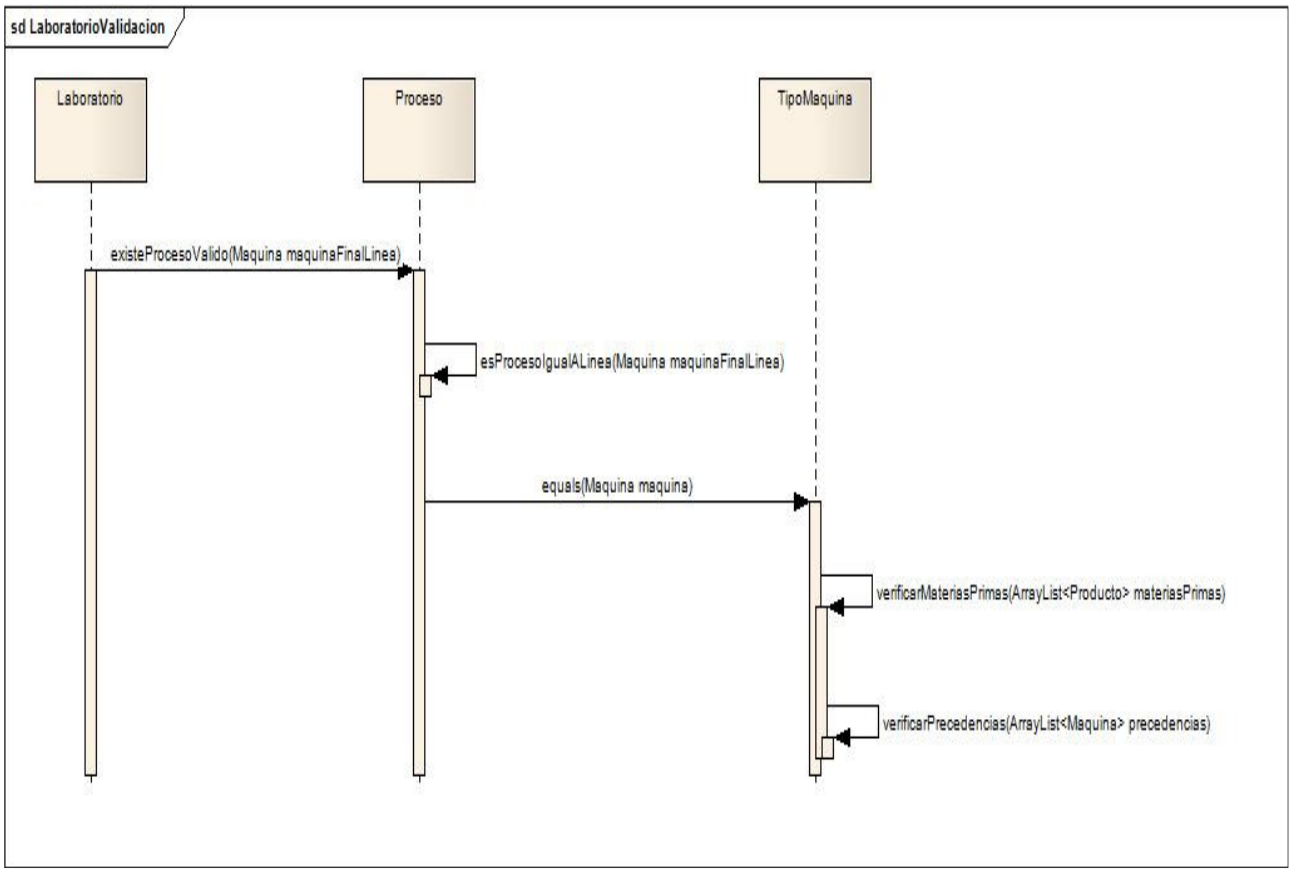


Diagrama de Secuencia Laboratorio – Validación



Fabrica - Compra, Venta y Alquiler

El problema a solucionar era como comprar, vender y alquilar nuevas fábricas. Para eso se pensó la idea de tener fábricas creadas al comienzo de la partida y que estas sean asignadas o removidas al jugador.

El jugador dado su dinero actual seleccionará la fábrica que desea comprar o alquilar de la lista de fábricas disponibles y luego podrá venderla.

Las fábricas tienen un costo de compra, un costo de alquiler por mes y los metros cuadrados que ocupa.

Si el jugador desea comprar una fábrica, se verificará si cuenta con el dinero suficiente. Se verificará que no tenga ya una fábrica adquirida y que la fábrica no haya sido adquirida por otro jugador. Si se cumplen las condiciones entonces podrá adquirir la fábrica y se le debitará el dinero correspondiente

Si el jugador desea alquilar una fábrica, no se verificará si tiene o no dinero ya que no es necesario poseerlo para alquilar, ya que el alquiler es gratuito. Se verificará que no tenga ya una fábrica adquirida y que la fábrica no haya sido adquirida por otro jugador. Si se cumplen las condiciones entonces podrá adquirir la fábrica y al final de cada mes se le debitará el dinero correspondiente al costo de alquiler.

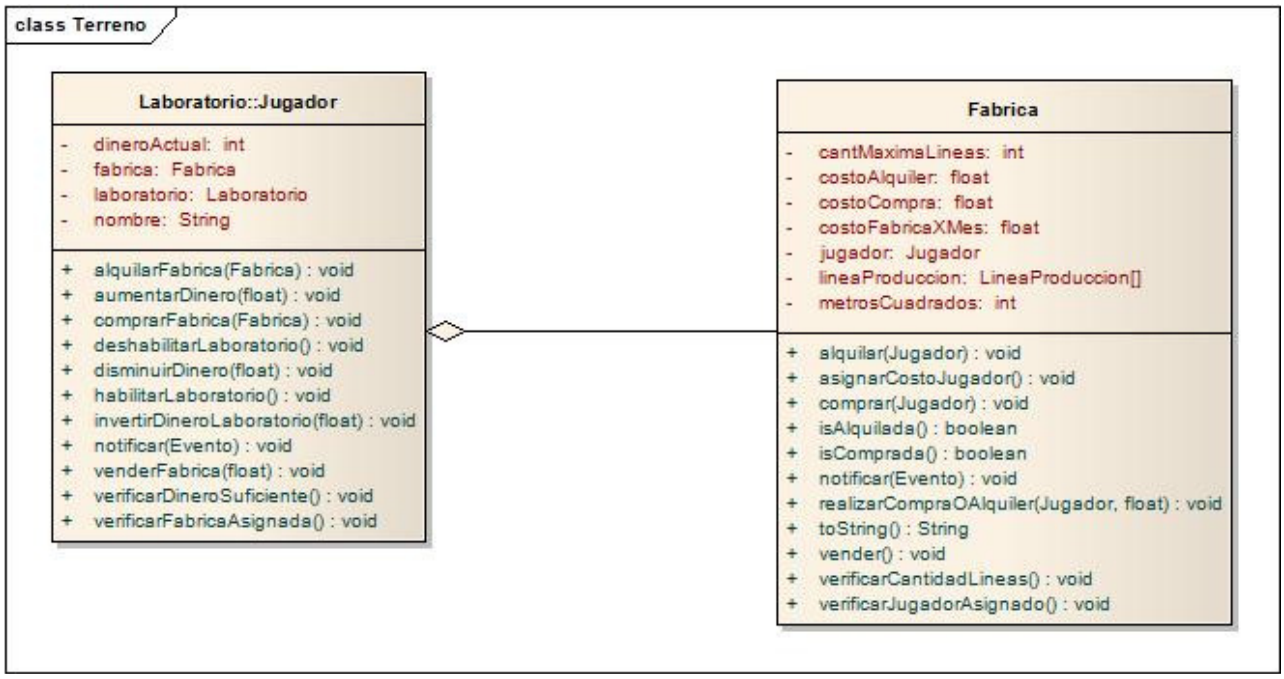
Tanto en la compra como el alquiler el jugador tendrá una referencia a la fábrica y la fábrica una referencia al jugador. La fábrica sabrá si fue alquilada o comprada y según eso por mes le debitará un costo al jugador, siendo de $0 en caso de ser comprada y del costo del alquiler en caso de ser alquilada.

Si el jugador desea vender una fábrica podrá hacerlo sin ningún problema, siempre y cuando tenga adquirida una fábrica. La venta es de una fábrica alquilada o comprada y según eso se le devolverá o no el costo de compra reducido en un 80% (en caso de alquiler no se le devuelve nada), más los un porcentaje de los costos de las máquinas que no están rotas en sus líneas de producción. El jugador dejará de referenciar a la fábrica y la fábrica dejará de referenciar al jugador, pasando a estar disponible para una compra o alquiler.

Las clases participantes son las siguientes:

- Jugador: Representa a un jugador, el cual tiene su dinero y toma decisiones a cerca de su fabrica.
- Fábrica: Representa a una fábrica particular, con un costo de alquiler, costo de compra, metros cuadrados y cantidad máxima de líneas de producción.

Diagrama de Clase Fábrica – Compra, Venta y Alquiler



Máquinas

Las máquinas son las encargadas de realizar las transformaciones de los productos, toman un conjunto de productos de su “entrada”, realizan una serie de validaciones y, si estas fueron superadas, procesan el producto y lo dejan en la “salida” listo para que pase a la máquina siguiente o al contenedor de productos terminados.

Dentro del contexto del juego se puede decir que un conjunto de n máquinas interconectadas conforman una linea de producción, pudiendo existir varias lineas por fábrica hasta un límite definido al momento de la creación de esta última.

Cada linea de producción tiene un costo que es proporcional al de las máquinas que la conforman.

Las máquinas son susceptibles de roturas, en este caso solo produciran desechos hasta que sean reparadas, la reparación tiene un costo que equivale a un porcentaje del precio de la máquina.

Relación con los elementos de la fábrica

La manera que tienen de conectarse con el resto de los elementos es la cinta de transporte, una cinta va desde la salida de una máquina (o desde una fuente) hacia la entrada de otra máquina (o a un contenedor).

La entrada de una máquina puede recibir materiales desde una o más cintas, en cambio las salidas solo pueden conectarse a una cinta transportadora. De esta manera se configura un grafo que recorre desde las fuentes hasta los contenedores pasando por una o más máquinas dependiende del producto a desarrollar.

Diagramas

A continuación se presenta una serie de diagramas que intentan mostrar desde varias vistas como ocurre la interacción arriba descripta.

En el primero tenemos una vista estática de las clases que componen la interacción, en el puede verse como la cinta se relaciona con la máquina a través de la entrada y salida.

El segundo diagrama muestra un instante dado en la vida de la fábrica, en el se han creado dos fuentes, tres máquinas y se realizaron las conexiones pertinentes a través

de las cintas transportadoras (Para simplificar el diagrama se omitieron las instancias de las entradas y salidas ya que no son relevantes).

El último diagrama modela los pasos que debe realizar un jugador para llegar a tener el modelo que se ve en el diagrama 2.

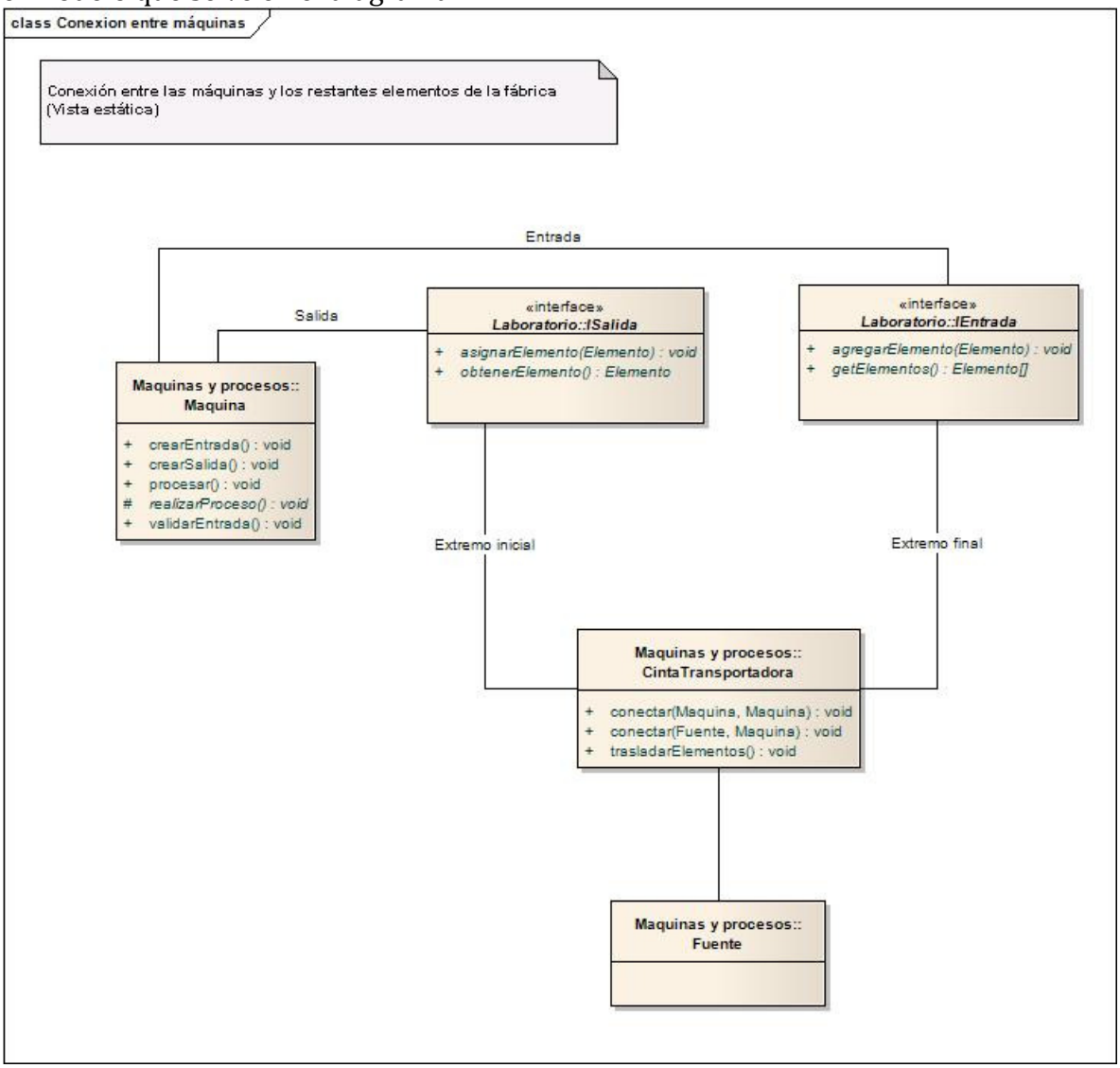


Diagrama 1 – Vista estática de la relación entre elementos de la fábrica.

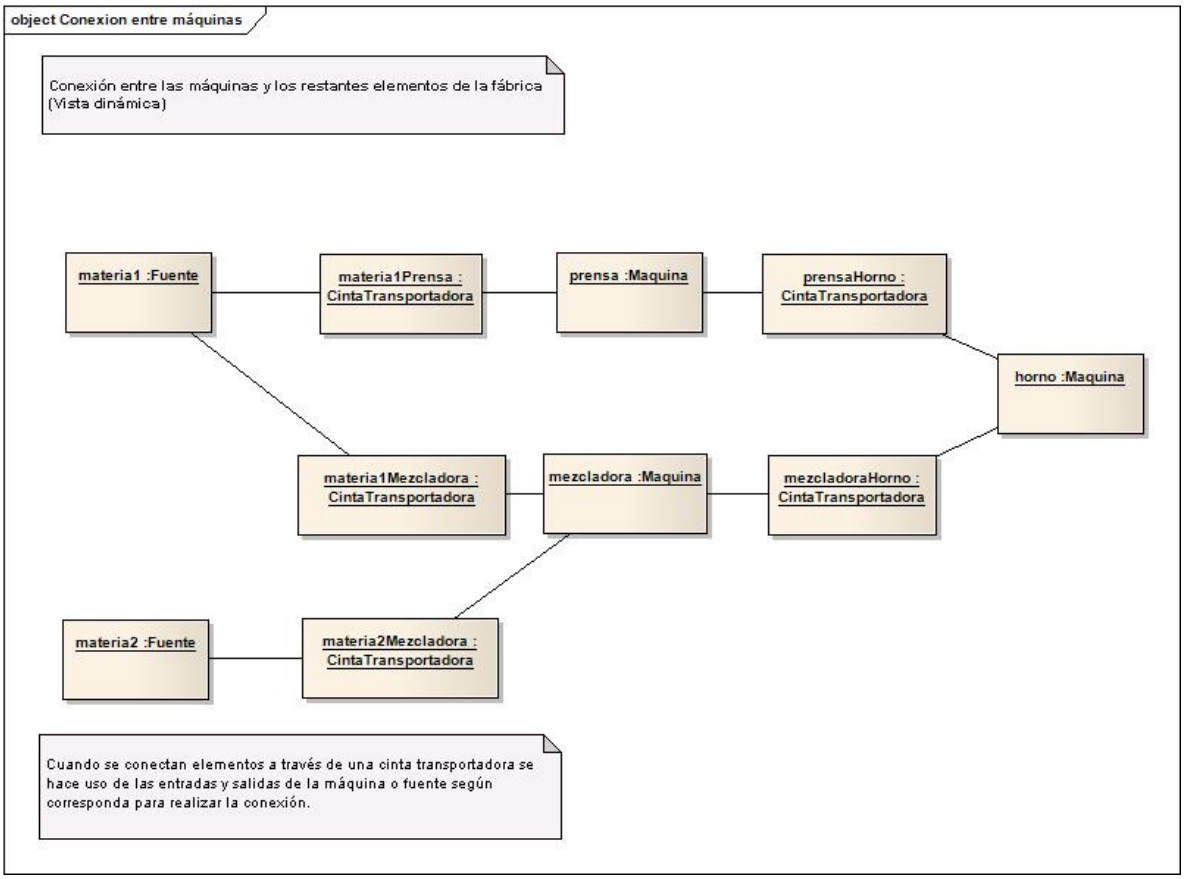


Diagrama 2 – Vista dinámica de un momento dado en la fábrica.

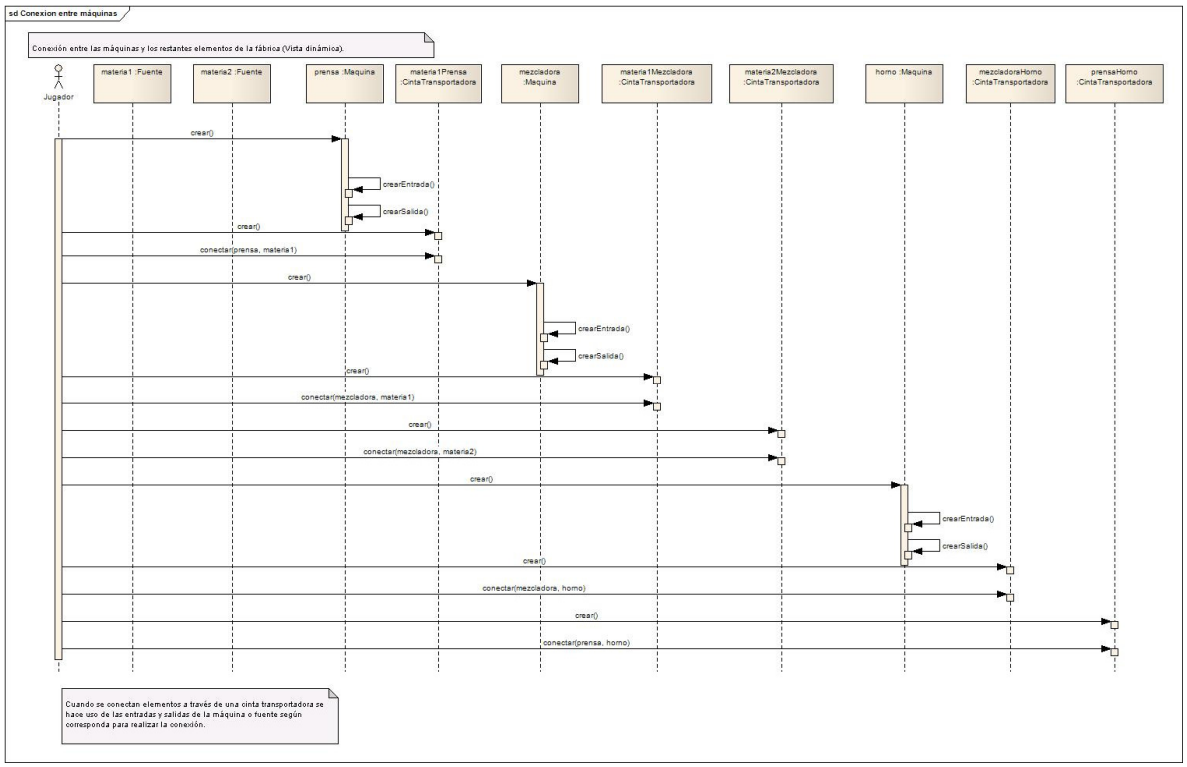


Diagrama 3 – Vista dinámica, pasos para obtener el modelo de objetos del diagrama 2

Relación con los productos

Una máquina toma una serie de productos y les realiza una transformación (proceso), esto se traduce en un cambio de estado del producto, quedando el nuevo producto disponible para ser procesado por la máquina siguiente en la linea, depositado en el contenedor si llegó a un estado final, descartado como desecho si se trata de un producto que no es conocido por el validador (Laboratorio) o descartado por defectuoso en caso de que se produzca una "falla" en la máquina.

Diagramas

En los siguientes diagramas se muestra el modelo estático de relación entre la máquina y el producto y una vista dinámica en la que se ve como una serie de productos interactúan con las máquinas.

En el diagrama de objetos se muestra como las fuentes crean (proveen) de productos que, a través de las cintas, llegan a las máquinas, estas realizan su procesamiento generando un producto nuevo, en el caso de las mezclas de dos o más productos, o alterando el estado del producto recibido.

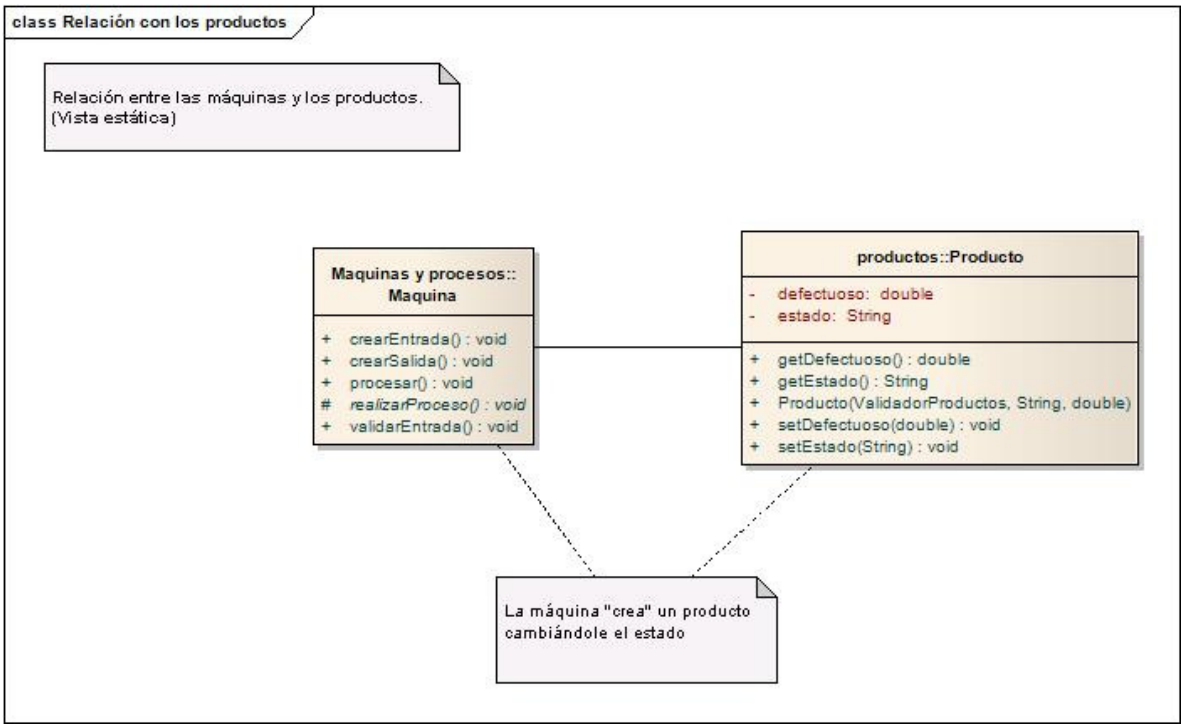


Diagrama 1 – Vista estática de la relación entre los productos y las máquinas.

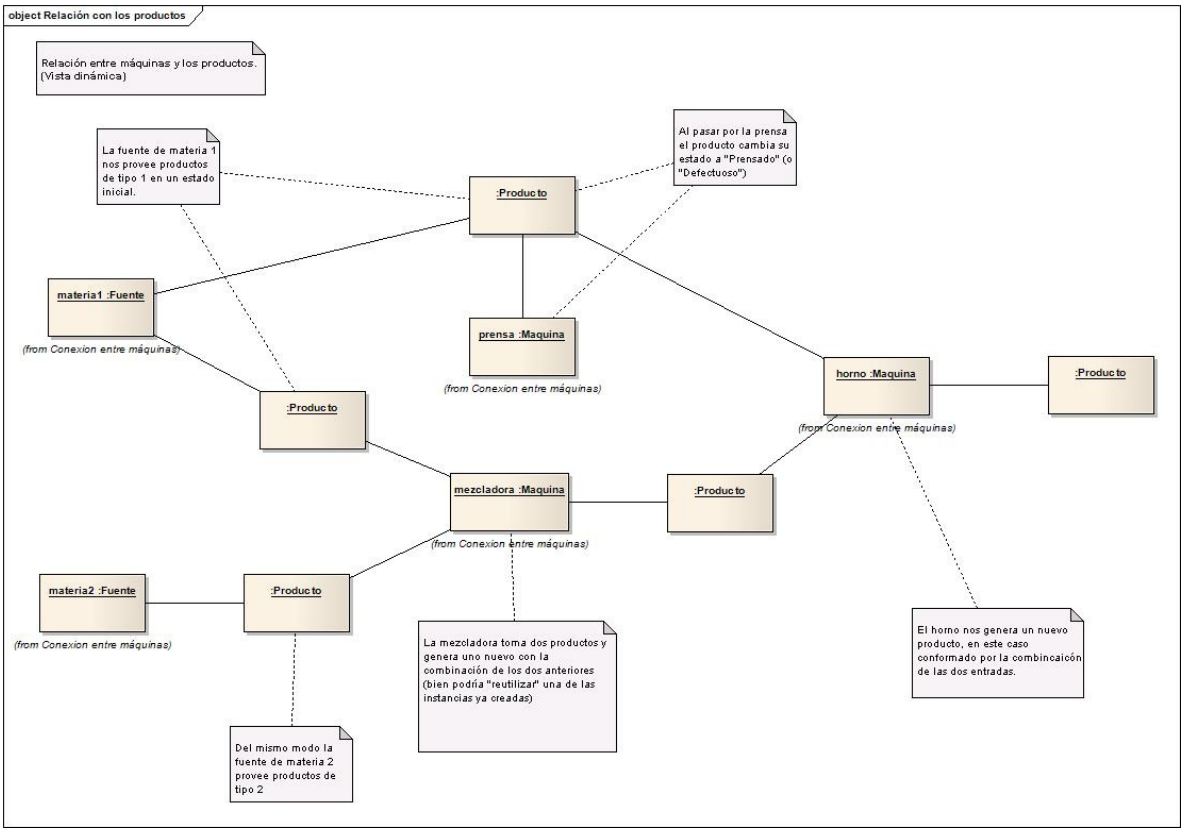


Diagrama 2 – Vista dinámica del flujo de productos entre las máquinas.

Calendario Virtual – Notificación de eventos

Al pasar un día del Calendario virtual, las partes de la línea de producción deben realizar tareas predeterminadas. Al pasar una semana, los precios de la Fábrica deben actualizarse. Al pasar un día, también debe incrementarse la inversión en el Laboratorio, para lo cual el Jugador debe transferirle dinero. Al mismo tiempo, el Laboratorio debe liberar a diario los procesos para los cuales se haya acumulado capital suficiente. Y en cuanto a la interfaz, no debe poder editarse la fábrica de no estar pausado el paso del tiempo.

El Calendario lleva cuenta del pasar de los días y las semanas, determinando cuándo ocurren estos eventos pero, para realizar sus tareas, los demás objetos del dominio deberían consultarlo constantemente para mantener el sincronismo. La solución a este problema es invertir el control: el Calendario comunicará dichos eventos a quienes quieran suscribirse, que deberán implementar la interfaz Sincronizado, y que ejecutarán las acciones que consideren necesarias cuando el Calendario decida notificarlos.

La responsabilidad de qué hacer cada día quedará entonces delegada en los objetos del modelo. No es de la incumbencia del Calendario qué procesos deben llevarse a cabo, sólo notificar el paso del tiempo y eventos propios de su funcionamiento.

Diagrama de Clases

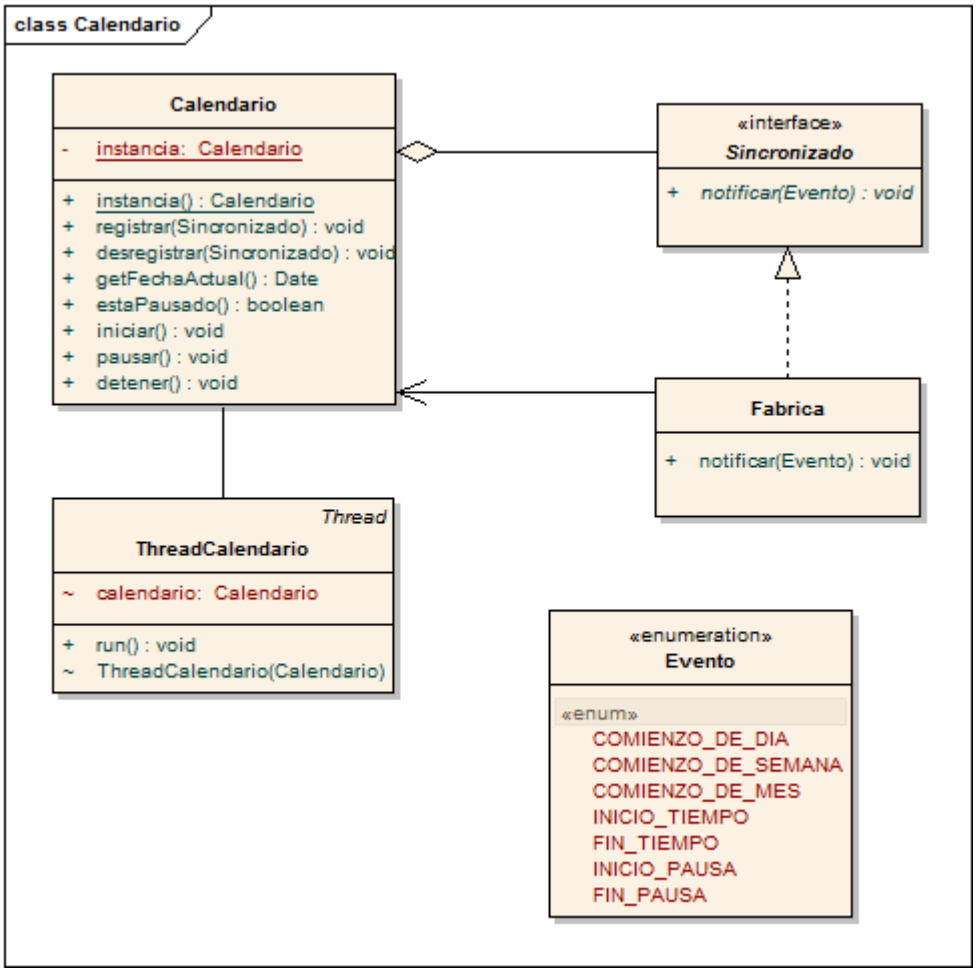
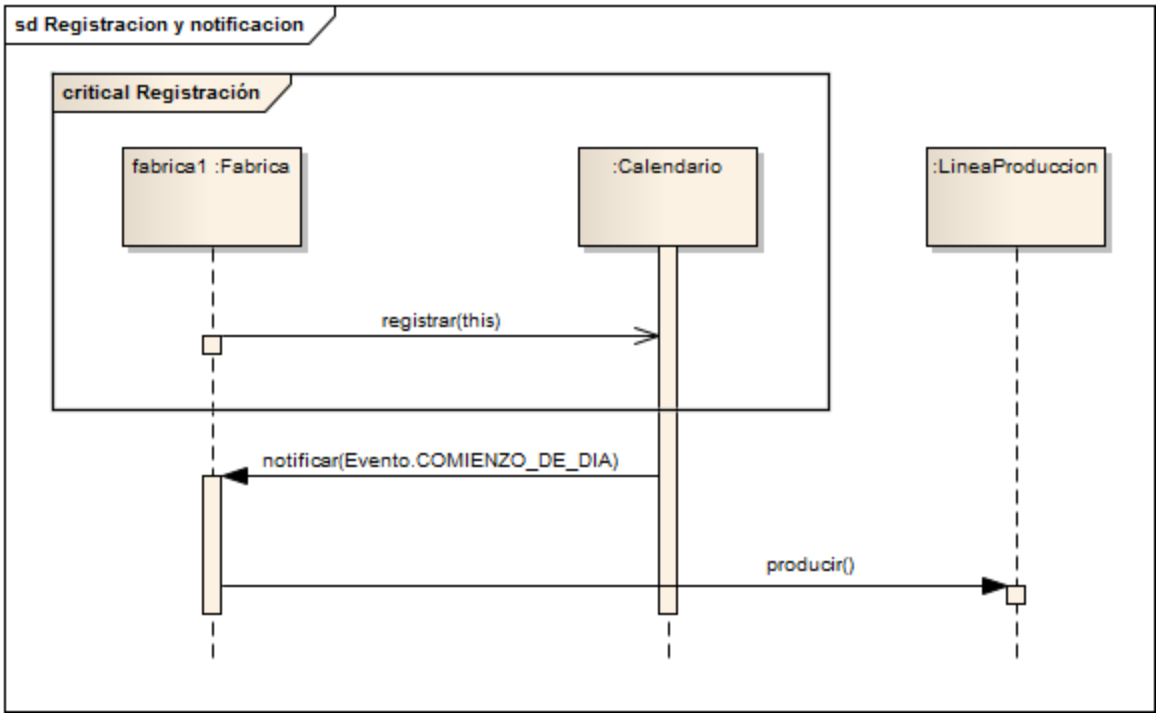


Diagrama de Secuencia: registración y notificación



Interfaz Gráfica – Diseño de Fábrica

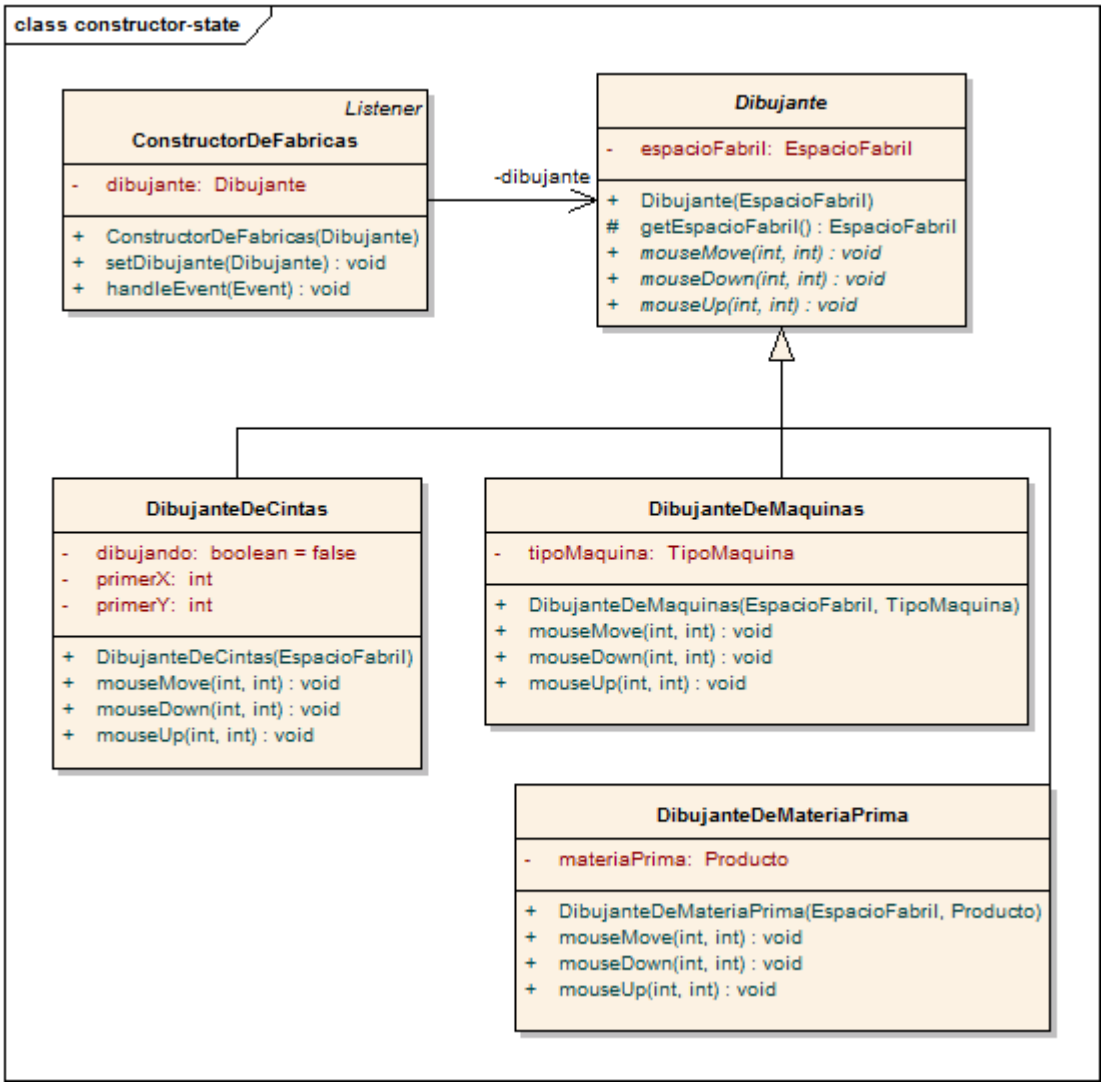
Para el dibujo del área de una fábrica se utilizó el widget Canvas de SWT. El mismo es notificado de los eventos del mouse, y a partir de estos deben crearse los componentes de una fábrica. El problema que encontramos es que estos eventos corresponderán a distintas acciones según qué se esté haciendo: creando máquinas, conectándolas con cintas, eliminando máquinas, etc. Las acciones controladas mediante el uso del mouse se eligen a través de controles de la interfaz como botones y listas.

La solución se implementa aplicando el patrón State: una instancia de la clase ConstructorDeFabricas cambia su comportamiento cuando el usuario decide cambiar de acción. Los eventos que provocarán los cambios de estado del constructor son la selección de un tipo de máquina a crear, la selección de la opción de creación de cintas transportadoras, la de la opción de vender máquinas, etc.

Esto hace extensible el comportamiento que inducen los eventos del mouse, e impide que se determine mediante variables de estado que complicarían el código y lo harían menos claro, y definitivamente menos mantenible. Con la solución propuesta, la clase ConstructorDeFabricas queda cerrada ante cambios, y sin embargo pueden agregarse nuevos estados que realicen acciones diferentes sobre el espacio de la fábrica. La ventaja en extensibilidad de este diseño fue aprovechada durante el desarrollo, ya que los distintos comportamientos fueron introducidos a medida que el modelo los soportaba, simplemente agregando especializaciones de la clase Instalador, cuyas instancias hacen las veces de estado y proveen el funcionamiento adecuado en respuesta a los eventos de la interfaz.

El estado de un constructor es una especialización de la clase abstracta Dibujante; el nombre de dicha clase aún se encuentra en revisión.

Diagrama de Clases


class constructor-state
Listener
ConstructorDeFabricas
- dibujante: Dibujante
+ ConstructorDeFabricas(Dibujante)
+ setDibujante(Dibujante) : void
+ handleEvent(Event) : void
-dibujante
Dibujante
- espacioFabril: EspacioFabril
+ Dibujante(EspacioFabril)
# getEspacioFabril() : EspacioFabril
+ mouseMove(int, int) : void
+ mouseDown(int, int) : void
+ mouseUp(int, int) : void
DibujanteDeCintas
- dibujando: boolean = false
- primerX: int
- primerY: int
+ DibujanteDeCintas(EspacioFabril)
+ mouseMove(int, int) : void
+ mouseDown(int, int) : void
+ mouseUp(int, int) : void
DibujanteDeMaquinas
- tipoMaquina: TipoMaquina
+ DibujanteDeMaquinas(EspacioFabril, TipoMaquina)
+ mouseMove(int, int) : void
+ mouseDown(int, int) : void
+ mouseUp(int, int) : void
DibujanteDeMateriaPrima
- materiaPrima: Producto
+ DibujanteDeMateriaPrima(EspacioFabril, Producto)
+ mouseMove(int, int) : void
+ mouseDown(int, int) : void
+ mouseUp(int, int) : void

Interfaz Gráfica

Para la jerarquía de GUI, utilizamos básicamente un Menú Principal y varios menús accesorios.

La idea original fue volcar la información en la interfaz de manera que el jugador pudiera acceder de forma simple y sencilla a los datos que requiera (Fecha calendario, Dinero, etc.). De esa forma lo que se decidió fue encapsular en partes la información relevante a esa parte.

Si bien aún es posible llegar a un mejor diseño, la interfaz cumple con todo lo expuesto anteriormente, dado que la misma nos permite utilizar de forma sencilla todos los componentes.

Diagrama de Clases

Producto

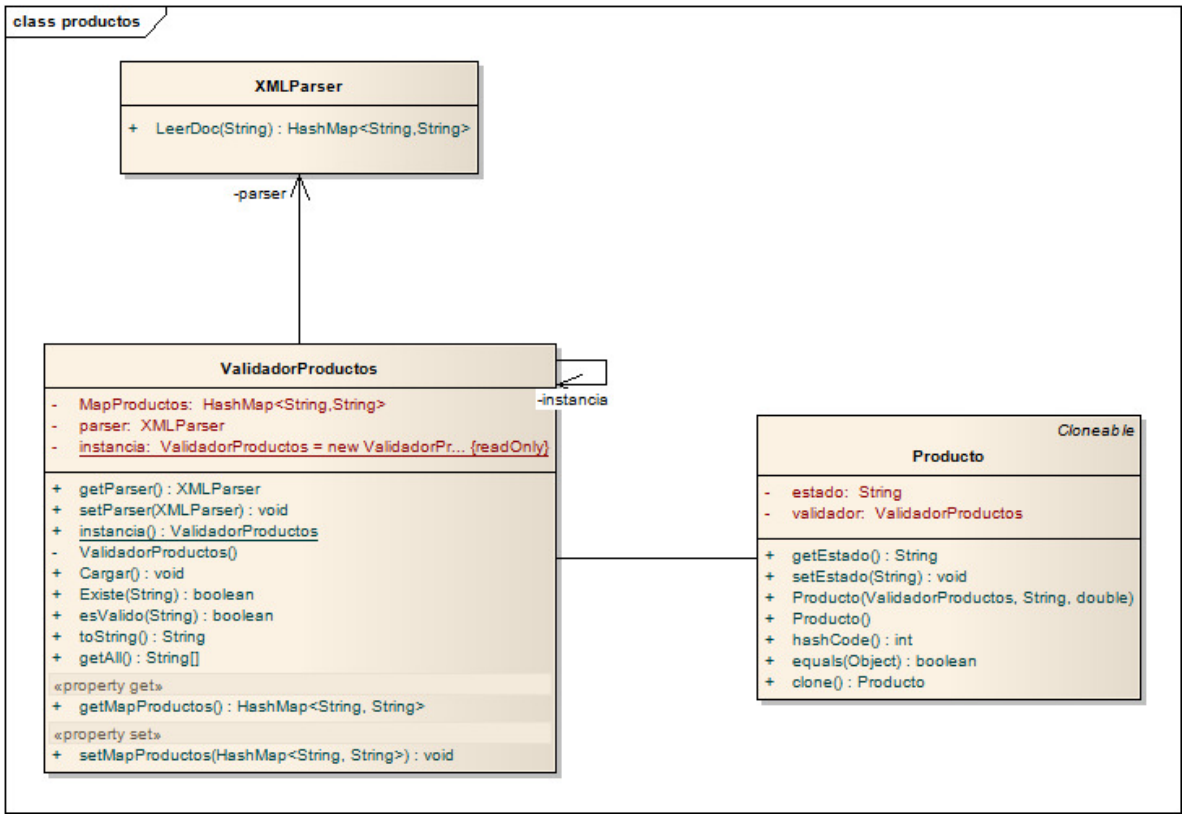
Para la jerarquía de productos, hemos decidido de común acuerdo la opción de contar con una clase producto que simbolice a todos los productos. El motivo se esta decisión tiene origen en lo expresado en el enunciado al decir “Las maneras de producir productos deben poder ser configurables mediante un xml. El laboratorio "habilita" productos existentes en el xml, considerando aquellos productos con capital de inversion cero como disponibles desde el inicio de la partida.”

Aquí podemos observar que los mecanismos de producción de un producto particular deben ser configurables mediante un archivo y los productos existentes serán habilitados mediante otro archivo. Tomando esta parte y viendo que los productos en si mismos no poseen una amplia funcionalidad (solo se crean y luego pasan a ser vendidos o utilizados como más materia prima) es que creimos conveniente tener una única clase “Producto”. La misma, para instanciar un producto, valida la existencia del tipo de producto dentro de un HashMap (ValidadorProducto) que tiene como función proveer de información a la clase producto sobre los tipos existentes. En caso de no existir el producto, el mismo se creará como desecho, en caso contrario, se validará que la creación de dicha instancia esté dentro de los parámetros de corrección del proceso de la fábrica, teniendo asociada, una tasa de defecto. Si el producto “nace” fallado, quedará con estado defectuoso y sino, quedará con el estado con el que se lo intentó crearlo.

Este modelo nos permite la libertad de poder agregar productos, siempre y cuando lleguen al HashMap, y para ello, podemos contar con el archivo de configuración xml de productos.

A continuación, ajuntamos un diagrama de clases de Producto, mostrando su relación con el validador y el Parser XML, a fines de mostrar como se llega a crear cualquier producto.

Diagrama Clase Producto



Patrones de diseño utilizados

Durante el diseño del juego fue surgiendo la necesidad de resolver problemas comunes, en esos casos se intentó aplicar los patrones de diseño vistos en clase, a continuación se listan algunos de los patrones utilizados junto con una descripción de su necesidad y su implementación.

Singleton

Determinados objetos debían ser compartidos por varios componentes de la aplicación, por ejemplo el calendario que controla el tiempo, en estos casos se decidió modificar el objeto para no permitir múltiples instancias del mismo.

Prototype

El juego cuenta con varios tipos de máquina y de cada tipo se pueden colocar la cantidad deseada en el escenario, dado que todas las máquinas son, en esencia, iguales, se decidió que el tipo de máquina sea un prototipo que genera máquinas.

Bridge

Durante el diseño del esquema de máquinas decidimos que cada tipo de máquina podía llegar a compararse con otro de diferentes maneras y establecimos un bridge entre los tipos de máquina y los comparadores. En este caso el patrón no fue aprovechado dado que no fue necesario comparar máquinas de distintas maneras.

Observer

Al modelar la aplicación nos encontramos con la necesidad de notificar a varios objetos de determinados cambios, por ejemplo el paso del tiempo, ya que cada día (o cada fin de mes) se ejecutan ciertas operaciones, para estos casos establecimos un esquema de suscripción y notificación, aquellos objetos que deben hacer tareas cada un determinado tiempo se suscriben al calendario quien los notifica cada vez que pasa un día o mes.

Iterator

Un laboratorio tiene muchos procesos y cada proceso cuenta con varios pasos, para recorrer esta estructura y verificar si un conjunto de máquinas es una linea válida se utilizó un iterador.

State

La clase ConstructorDeFabricas tiene un estado representado por una especialización de la clase abstracta Instalador, que le permite modificar su comportamiento accediendo, mediante la interfaz de esta última, a las distintas implementaciones de sus operaciones.

Tempate method

Las máquinas están planteadas siguiendo este patrón donde una máquina abstracta delinea las tareas generales (obtener materia, validar, procesar, depositar materia) y cada máquina concreta redefine los pasos necesarios.

Composite/Decorator

La UI de la aplicación, guiada por el framework elegido, se llevó a cabo componiendo y decorando objetos.

Probablemente hayamos utilizado otros patrones que en este momento no recordamos o que no reconocimos como tales, también es posible que algunos de nuestros problemas se podrían haber resuelto de mejor manera aplicando algún otro.

Minutas de reunión

Fecha de la reunión: 13/05/2010

Participantes

Federico Diaz	Ayudante
Diego García Jaime Santiago N. Risaro Sesar Gustavo Ariel Meller Esteban Ignacio Invernizzi	**Grupo 3**

Temas Tratados

Consultas generales, presentadas por el grupo, sobre el dominio del problema.
Definición de las entregas de diseño a presentar al ayudante en las próximas reuniones:
Centrarse en un problema del dominio al presentar consultas. Es conveniente que las primeras consultas se centren en los problemas más críticos del dominio.
Presentar un diseño sobre dicho problema. El diagrama de clases debe ser claro en el sentido que se tiene que centrar solo en ese problema en cuestión (solo muestra las asociaciones, atributos y métodos relacionados). Dependiendo de la complejidad del problema podrían ser varios diagramas de clase.
Validación del diseño. Se deben presentar otros artefactos que validen el diseño. Estos pueden ser: Listados de requerimientos, supuestos, diagramas UML dinámicos, codificación de la solución y test unitarios. Los diagramas de clases no deben tener errores en cuanto a métodos o atributos que aparezcan en un artefacto de validación y no en los diagramas de clase (igualmente si aparecen con distinto nombre), deben ser coherentes y completos en ese sentido.

Compromisos

El grupo deberá asignarse al menos un problema de diseño para cada integrante o tres problemas de diseño y el cuarto integrante el listado de requerimientos.
Presentaran la solución de diseño al problema. Esto es: diagramas de clase con la descripción del problema de dominio que se modelo, descripción de las clases (responsabilidades, colaboraciones). Presentaran artefactos que validen el diseño (ej: diagramas de secuencia describiendo los escenarios del problema, listado de requerimientos o supuestos sobre el dominio que muestren una completitud de la solución, diagramas de estado etc.). Codificaran el diseño y test unitarios.
El grupo entregara el jueves 21/05 un informe que consistirá en un documento (carátula, índice etc.) que recopile el trabajo descripto en los puntos anteriores.

Fecha de la reunión: 20/05/2010

Participantes

Federico Diaz	Ayudante
Diego García Jaime Santiago N. Risaro Sesar Gustavo Ariel Meller Esteban Ignacio Invernizzi	**Grupo 3**

Compromisos asumidos por los integrantes del grupo

A continuación se enumeran los compromisos asignados a cada integrante del grupo. Cada compromiso es sobre un problema del dominio.

Diego: Diseño del modelo de producto. Diseño del modelo de creación/creaciones de producto.

Santiago: Diseño del modelo de máquina. Diseño del modelo de maquina relacionando las conexiones con otros elementos de la fabrica (Cintas, Entradas Salidas). Diseño del modelo de máquina en relación con los productos.

Gustavo: Diseño del laboratorio en cuanto a lógica de inversiones. Diseño de validaciones del laboratorio frente a distintas líneas de producción.

Esteban: Diseño del modelo del avance temporal del juego, cómo afecta a los distintos objetos del dominio.

Cada alumno deberá realizar su entrega cumpliendo con los siguientes requisitos:

Diagramas de clase orientados al problema de diseño con su titulo, descripción del problema, descripción de las clases (responsabilidades, relaciones con otras clases). Se debe llegar a una solución completa del problema (por eso se lo divide en la mayor medida posible). Se debe verificar que el diseño alcanza la solución completa, para esto se pueden presentar: diagramas de secuencia con los distintos escenarios (u otros diagramas UML), hipótesis, trazabilidad con el listado de requerimientos. Se debe entregar codificada la solución, con un set de test unitarios.

Compromisos asumidos por el ayudante

- Corrección del listado de requerimientos entregado.

- Confección del calendario de entregas del trabajo práctico.

Fecha de la reunión: 03/06/2010

Participantes

Federico Diaz	Ayudante
Diego García Jaime Santiago N. Risaro Sesar Gustavo Ariel Meller Esteban Ignacio Invernizzi	**Grupo 3**

Temas tratados y compromisos

Se realizo una demo del juego. La demo era reducida en cuanto a la funcionalidad del juego, pero se pudo definir lo faltante dejándolo como pendiente para las próximas entregas.

Se acordó el siguiente calendario de entregas y correcciones en la etapa final del trabajo practico:

Lunes 7 de Junio Entrega de informe final, preliminar.

Jueves 10 de Junio Demo del juego. Código fuente preparado para revisión.

Lunes 14 de Junio Devolución de corrección: Informe + Diseño + Código. Negociación de la nota final.

Jueves 17 de Junio Entrega final: Informe Final + CD (Fuentes + txt de instalación)

El informe final preliminar tiene que tener el formato de la entrega final, secciones ejemplo:

Carátula + Tabla de contenidos + Enfoque + Descripción de la arquitectura General + Secciones sobre los problemas del dominio (descripción del problema, diagramas de clases, descripción de los conceptos establecidos en el diagrama, descripción de escenarios con los diagramas UML de secuencia, diagramas de estados, si utilizaron patrones como los identificaron o se desprendieron del diseño. Agreguen las ventajas y desventajas de cada uno, traten de validarlo.) + Apéndice (podrían poner un

apartado para el análisis que realizaron y otro en el cual describan todas las herramientas utilizadas, desde repositorio hasta las distintas librerías).

La entrega preliminar del lunes puede ser en el horario de las 21 hs del lunes (eventualmente se pueden consultar sobre el trabajo practico) o puede entregarse electrónicamente. Si deciden juntarse el lunes a las 21 hs enviar e – mail para confirmar la reunión.

La demo del jueves 10 de Junio tiene que ser una demo en la que ya se pueda Jugar al Juego.

En la devolución de la corrección del Lunes 14 de Junio se negocia la nota y puntos a corregir para la entrega final.

Conclusiones

Luego de haber realizado el trabajo nos quedan las siguientes conclusiones.

Creemos que el tiempo utilizado para el trabajo fue escaso lo cual hizo que tengamos menos tiempo para debatir y decidir en la parte de diseño llegando a diseñar cosas consensuadas entre los integrantes que, en el momento de la implementación, hubo que cambiarlas sin tanto debate.

Pensamos que quizás se debatieron temas que en su momento se pensaban como más importantes y que a la larga terminaron siendo irrelevantes, apareciendo otros de mayor relevancia y llevándose mayor tiempo de análisis de lo pensado.

Las decisiones fueron tomadas respetando las opiniones de los integrantes, logrando un rápido consenso y colaboración entre los integrantes, los cuales fuimos aportando ideas y sugerencias a los distintos problemas, a pesar de que cada integrante se haya dedicado exclusivamente a una serie de problemas en particular.

La coordinación con el ayudante creemos que fue buena y cada uno supo lo que debía hacer para la siguiente clase, teniendo un calendario claro y una idea de lo que se tenía que llevar en cada ocasión. Esto hizo que todo pudiera ser más organizado y las clases fuesen más productivas.

Como último punto sugerimos a la cátedra, para que lo puedan tomar a cuenta en un futuro si es que lo desean, que el TP se entregue con una mayor anticipación de tiempo, para que se pueda trabajar mejor y con más tiempo, ya que creemos que se podrían lograr mejores resultados.