SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Tecnológico Nacional de México
Coordinación Sectorial Académica
Dirección de Estudios de Posgrado e Investigación

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Centro Nacional de Investigación y Desarrollo Tecnológico

Subdirección Académica

Departamento de Ciencias Computacionales

TESIS DE MAESTRÍA EN CIENCIAS

Herramienta de Soporte al Modelado de Software por Combinación de Patrones de Diseño

presentado por

Ing. Manuel Bernardo Ibáñez Ocampo

como requisito para la obtención del grado de
Maestro en Ciencias de la Computación

Director de tesis
Dr. René Santaolaya Salgado

Codirectora de tesis
Dra. Olivia Graciela Fragoso Díaz

Cuernavaca, Morelos, México. Enero de 2016.

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Mor., 18/enero/2016
OFICIO No. DCC/017/2016

Asunto: Aceptación de documento de tesis

C. DR. GERARDO V. GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial del **C. Manuel Bernardo Ibáñez Ocampo**, con número de control M13CE056, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis profesional titulado "**Herramienta de soporte al modelado de software por combinación de patrones de diseño**" y hemos encontrado que se han realizado todas las correcciones y observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

DIRECTOR DE TESIS

Dr. René Santaolaya Salgado
Doctor en Ciencias de la Computación
4454821

CO-DIRECTORA DE TESIS

Dra. Olivia Graciela Fragoso Díaz
Doctora en Ciencias en Ciencias de la Computación
7420199

REVISOR 1

M.C. Humberto Hernández García
Maestro en Ciencias con Especialidad en Sistemas Computacionales
7573641

REVISOR 2

Dr. Juan Carlos Rojas Pérez
Doctor en Ciencias en Ciencias de la Computación
6099372

REVISOR 3

Dr. Moisés González García
Doctor en Ciencias en la Especialidad de Ingeniería Eléctrica
7501724

C.p. Lic. Guadalupe Garrido Rivera - Jefa del Departamento de Servicios Escolares.
Estudiante
Expediente

AMR/lmz

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Mor., 19 de enero de 2016
OFICIO No. SAC/035/2016

Asunto: Autorización de impresión de tesis

C. MANUEL BERNARDO IBÁÑEZ OCAMPO
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS
DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Herramienta de soporte al modelado de software por combinación de patrones de diseño"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

ATENTAMENTE
"CONOCIMIENTO Y TECNOLOGÍA AL SERVICIO DE MÉXICO"

CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO SUBDIRECCIÓN ACADÉMICA

DR. GERARDO VICENTE GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO

C.p. Lic. Guadalupe Garrido Rivera.- Jefa del Departamento de Servicios Escolares.
Expediente

GVGR/mcr

Agradecimientos

A mi familia, mis padres Bernardita de Lourdes Ocampo Sandoval y Manuel Ibáñez Rivero y mi hermano Víctor Manuel Ibáñez Ocampo, por guiarme y apoyarme para llegar a este grado.

A mi pareja Laura Corazón Rebolledo Castañeda, por su apoyo incondicional e incontables desveladas apoyándome a realizar trabajos.

Al Centro Nacional de Investigación y Desarrollo Tecnológico por haberme dado la oportunidad de formar parte de su programa de posgrado.

Al Dr. René Santaolaya Salgado director de tesis y la Dra. Olivia Graciela Fragoso Díaz codirectora de tesis por su apoyo y tutela durante todo el proceso de desarrollo de esta tesis.

Al comité de revisores de esta tesis, que está conformada por el M.C. Humberto Hernández García, el Dr. Juan Carlos Rojas Pérez y el Dr. Moisés González García por su tiempo usado en revisar y mejorar esta investigación.

Muchas Gracias

Tabla de contenido

Lista de Figuras

Lista tablas

Resumen

Los patrones de diseño son micro-arquitecturas de software que brindan soluciones genéricas probadas que se aplican para soluciones específicas bajo diferentes. Cada uno tiene diferente intención y consecuencias, algunos son parecidos en estructura y se necesita experiencia por parte del desarrollador para saber cuándo y dónde aplicarlos de forma correcta.

El objetivo de esta tesis es ampliar la funcionalidad del Sistema Verificador de Patrones (SiVerPaT), que originalmente solo tiene las reglas formales de *Iterator*, *Composite* y *Observer* y sus reglas de combinaciones, para implementar las reglas de combinación desarrolladas en la tesis *Reglas para la Combinación de Patrones de Diseño*, para la verificación de la correcta combinación de los patrones *Strategy*, *Composite*, *Factory Method* y *Template Method*. El desarrollo consiste de la construcción de un intérprete con la capacidad de de cargar un archivo de texto plano que contiene las reglas formales de un patrón de diseño o sus reglas de ccombinación y traducirlo a un archivo en formato XML, el cual es reconocido por la herramienta SiVerPat. En las pruebas se demuestra que el intérprete traduce de forma correcta las reglas formales y son cargadas en la herramienta SiVerPat de manera correcta, demostrando que el intérprete desarrollado produce archivos XML que son reconocidos por la herramienta SiVerPat.

Abstract

Design patterns are software micro-architecture that provides tested generic solutions that apply to specific solutions under different contexts. Each has different intentions and consequences; some are similar in structure and experience necessary for the developer to know when and where to apply them correctly.

The aim of this thesis is to extend the functionality of the System Checker Pattern (*SiVerPat*), that originally only it has the formal rules of the pattern Iterator, Composite and Observer and their combination rules, to implement the combination rules developed in the thesis *Reglas para la Combinación de Patrones de Diseño*, for verification of the correct combination of patterns Strategy, Composite, Factory Method y Template Method. The development consists of building an interpreter with the ability to load a plain text file containing the formal rules of a design pattern or their combination rules and translating to a file in XML format, which is recognized by the tool *SiVerPat*. In tests it demonstrated that the interpreter translates correctly formal rules and they are loaded into the tool *SiVerPat* in a correct way, showing that the developed interpreter produces XML files that are recognized by the tool *SiVerPat*.

CAPÍTULO 1. INTRODUCCIÓN

1.1. Introducción

El reuso aumenta la productividad de los desarrolladores pero no garantiza el aumento en la calidad del software. Es más sencillo localizar y corregir problemas en las primeras etapas de desarrollo que cuando el sistema ya está implementado. Una de estas etapas es la de diseño. El diseño indica cómo será construido el software, cómo se comportará y qué necesitará, antes de que sea implementado. Un mal diseño o no realizar la etapa de diseño en el desarrollo de software pueden inducir a defectos en el producto final.

Los principios de diseño orientado a objetos establecen una guía para las mejores prácticas de desarrollo de software de buena calidad. No aplicarlos tiene consecuencias, tales como: fragilidad, rigidez, impredecibilidad e inamovilidad del software. Si el software es frágil, cambios en su estructura hacen que deje de funcionar de forma correcta; la rigidez del software no permite cambios a la funcionalidad original del software, y no es posible ampliar su funcionalidad sin modificar su definición actual; lo impredecible del software se refiere a que un defecto puede propagarse a través del código sin determinar con certeza el alcance del defecto; el software inamovible impide su reuso y el de sus componentes en nuevas aplicaciones.

Otros elementos que ayuda a obtener software de buena calidad son los patrones de diseño. Christopher Alexander fue la primera persona en desarrollar y aplicar estos elementos. En su libro de arquitectura define el término como, "*Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez.*" (Gamma, 2003) La comunidad de Ingeniería de Software adoptó este concepto para poder reusar diseños que resuelven problemas que se presentan de forma recurrente. En el 2003, la "Banda de los Cuatro" presentaron un catálogo donde describen 23 patrones de diseño (Gamma, 2003). Además, normalmente las aplicaciones de software no se resuelven mediante la aplicación de un único patrón de diseño, sino por el uso combinado de varios de ellos. Los patrones de diseño se combinan de tal forma que el resultado de uno de ellos es la entrada de otro o de manera que se puedan usar como alternativa para algún método o en forma concurrente.

Aplicar correctamente el uso de patrones de diseño requiere de experiencia y conocimientos sobre éstos, combinarlos de manera correcta requiere aún más experiencia y creatividad. Estas cualidades es difícil que las tengan todos los integrantes de un equipo de desarrollo.

Existen pocas herramientas de modelado orientado a objetos que soportan el diseño de software con patrones. Las que usan patrones de diseño y las que permiten su combinación dejan al usuario (desarrollador) la decisión de cómo combinarlos estructuralmente y no verifican que se haya realizado de forma correcta tal combinación.

En esta tesis, la investigación está dirigida hacia la extensión de la herramienta SiVerPat de manufactura propietaria del CENIDET, que apoya el modelado orientado a objetos con soporte del uso de patrones de diseño y su combinación. La investigación agrega a SiVerPat la implementación de las reglas formales de combinación de Héctor Uriel Pérez Rojas (Perez, 2014) para combinar los patrones de diseño, *Strategy, Composite, Template Method, Factory Method y su combinación*.

Las principales investigaciones que anteceden a esta tesis son las desarrolladas por el grupo de Ingeniería de Software del Centro Nacional de Investigación y Desarrollo Tecnológico. Estas son el desarrollo del **Si**stema **Ver**ificador de **Pat**rones (SiVerPat) desarrollado por José Salazar Robles, que uso las reglas desarrolladas en la tesis de Sergio René Gonzáles Castañeda para la combinación y verificación de los patrones *Composite, Iterator y Observer*. Posteriormente en la tesis de Héctor Uriel Pérez Rojas se formularon las reglas formales de combinación de los patrones de diseño, *Strategy, Composite, Template Method, Factory Method, sin llegar a su implementación en alguna herramienta de modelado*. En estas reglas se incluyen Patrones que no son de combinación estructural, si no de comportamiento, tal es el caso del patrón "*Template Method*", estas combinaciones se explican en el capítulo 3.

El alcance de la herramienta SiVerPat es el siguiente:

- Ayuda a bosquejar el diseño de diagramas de clases creados por el usuario, basándose en las estructuras genéricas de los patrones de diseño *Composite, Iterator, Observer, Strategy, Template Methos y Factory Method*.
- Permite al usuario elegir los patrones de diseño que desea combinar y el sistema despliega el diagrama de clases con la combinación correcta de los patrones diseño participante.

Las clases del diagrama de clases creado son configurables, se puede cambiar su nombre, atributos o métodos. El usuario puede agregar más clases, siempre y cuando se respete la estructura del patrón, en el capítulo 3 se explica la estructura que debe ser respetada. El sistema proporciona la versión genérica de cada patrón y verifica que la combinación cumpla con las reglas establecidas en (Gonzáles, 2010)y (Pérez, 2014) estas reglas se explican con profundidad en el capítulo 3.

1.2. Planteamiento del problema

Debido a que algunos desarrolladores tienen poca experiencia, creatividad, imaginación y capacidad de abstracción, que no cuentan con herramientas que les ayude a combinar los patrones de diseño, ellos pueden obtener diseños arquitecturales frágiles, no-reusables, y rígidos a cambios o extensiones. Así, el problema es que se obtienen diseños con baja posibilidad de reuso y difícil de mantener.

1.3. Objetivos

1.3.1 Objetivo General

Agregar las reglas de combinación y verificación de los patrones de diseño *Strategy, Composite, Template Method, Factory Method* para ampliar la funcionalidad del Sistema Verificador de Patrones sin alterar las reglas de combinación y verificación establecidas en la herramienta.

1.3.2 Objetivos específicos

- Automatizar el proceso de conversión de las reglas gramaticales para SiVerPat.
- Convertir las reglas gramaticales *Strategy, Composite, Template Method, Factory Method* y sus combinaciones en archivos XML.
- Modificar SiVerPat para aceptar la carga de archivos independientes XML, que contengan las reglas gramaticales de patrones de diseño o combinaciones de estos.
- Modificar la interfaz de SiVerPat para agregar los botones que dibujen los diagramas de clase de los patrones de diseño y combinaciones de *Strategy, Composite, Template Method, Factory Method.*

1.4. Justificación

Los diseñadores que intentan aplicar patrones de diseño en aplicaciones grandes se les dificulta la combinación correcta de éstos. Para este fin se requiere de una herramienta que apoye al diseño que combine estos patrones. Antes de esta investigación *SiVerPat* sólo combinaba formalmente tres patrones de diseño, tomados de dos a dos, de los 23 disponibles en el catálogo de Gamma, pero se cuenta con las reglas de combinación de otros tres patrones de diseño desarrolladas por (Pérez, 2014), por lo que es posible extender el alcance de combinación de patrones de diseño a seis, tomados de dos patrones de diseño a la vez.

1.5. Alcance y limitaciones

La herramienta *SiVerPat*, antes de esta investigación, tenía la capacidad de modelar los diagramas de los patrones *Observer, Iterator y Composite*, así como verificar que estuvieran construidos y combinados de forma correcta. La arquitectura de clases obtenida era configurable. Es decir, se podían agregar clases, cambiar el nombre de éstas, incluir métodos y atributos en éllas; siempre y cuando se conservara la base estructural de cada patrón de diseño y sus combinaciones dos a dos, conforme las reglas formales de combinación establecidas.

Bajo el alcance de esta tesis se agregó la implementación de las reglas de combinación y verificación de los patrones de diseño, *Strategy, Composite, Template Method y Factory Method* sin perder las capacidades anteriores de *SiVerPat,* en el capítulo 6 se muestra el

proceso de integración de las nuevas reglas a *SiVerPat* y el capítulo 7 muestra el resultado de las pruebas usando estas reglas.

Sin embargo esta herramienta no es capaz de combinar patrones de los cuales no tenga las reglas definidas. Por ejemplo, no podrá combinar el patrón *Strategy* con el patrón *Observer* debido a que las reglas de su combinación no están desarrolladas. Además sólo es capaz de combinar dos patrones de diseño a la vez porque se necesitarían las reglas de combinación de tres o más patrones. Una secuencia de combinación de dos en dos, por ejemplo combinar *Strategy* con *Composite* y a la vez la *Composite* con el *Iterator* formando una secuencia de combinación de patrones de diseño requiere tener la regla que combine estos cuatro patrones de diseño.

Como parte de la solución se desarrolló un intérprete que permite traducir archivos TXT que contengan las reglas formales de los patrones de diseño, expresadas en gramática relacional, en archivos XML reconocidos por SiVerPat, en el capítulo 4 se presenta el desarrollo del interprete.

1.6 Estructura de la tesis

Esta tesis se encuentra dividida en los siguientes capítulos:

- En el capítulo 2 se revisan trabajos relacionados que deben cumplir criterios específicos y se realiza una comparativa con relación a esta tesis.

- En el capítulo 3 se definen todos los conceptos que se usan en esta tesis como la definición de un patrón de diseño; la forma de los patrones usados; ventajas y desventajas de cada patrón; las reglas formales de estos patrones y sus combinaciones; el algoritmo que se usa para verificar los patrones y las combinaciones.

- En el capítulo 4 se especifica el diseño del intérprete que permite traducir reglas formales para SiVerPat.

- En el capítulo 5 se especifica el desarrollo del intérprete, con ejemplo del archivo de regla formal traducido a XML

- En el capítulo 6 se explica cómo agrega una nueva regla a SiVerPat

- En el capítulo 7 resume el resultado de las pruebas realizadas a SiVerPat y el intérprete así como los casos de pruebas de cada uno.

- En el capítulo 8 se presentan las conclusiones de esta tesis y se mencionan trabajos futuros.

CAPÍTULO 2: ESTADO DEL ARTE Y TRABAJOS RELACIONADOS

Entre las investigaciones que han utilizado patrones de diseño, son pocas las que consideran sus posibles combinaciones. Algunas de estas investigaciones se enfocan en la etapa de codificación con el fin de generar código aplicando los patrones de diseño. Otras, desarrollan herramientas que sólo tienen la función específica de ofrecer sugerencias sobre el tipo de patrón que se debe aplicar para resolver un problema. Algunas otras buscan identificar patrones de diseño a través de técnicas de refactorización.

Los criterios de búsqueda para seleccionar los artículos relacionados con la temática de la presente investigación fueran los siguientes:

- La capacidad de la herramienta para modelar Patrones de diseño.
- La capacidad de la herramienta de verificación formal de que los Patrones de diseño sean modelados de forma correcta.
- La capacidad de modelar la combinación de patrones de diseño, mediante reglas formales.
- Los Patrones de diseño que considera la herramienta.
- La etapa del desarrollo de software en la que se hace la verificación formal.
- Las entradas y las salidas que tiene la herramienta para verificar la creación de patrones y su combinación.
- El lenguaje o notación usada para representar los Patrones de diseño.

El presente trabajo de investigación se enfoca en la mejora de una herramienta que permite al usuario elaborar diagramas de clases con patrones de diseño y verificar su creación de manera formal así como las combinaciones entre estos. En la literatura solo se reportan la posibilidad de combinar patrones de diseño, y una pequeña sugerencia para realizarlo pero al no realizarse de manera formal, la combinación es propensa a fallas. La mayoría de las investigaciones encontradas, solo mencionan el uso de patrones de diseño y las que permiten la combinación no siguen ninguna regla.

En ausencia a un estado del arte específico en mi tema, se revisaron las siguientes investigaciones. Las seis primeras investigaciones que se describen, sólo mencionan el uso de patrones de diseño pero no los combinan de manera formal. Las últimas cinco verifican que los patrones de diseño se creen de forma correcta y realizan algún tipo de combinación de patrones.

Al final de este capítulo se presenta una tabla comparativa entre las aportaciones que se reportaron en este capítulo.

2.1 Automated verification of design patterns: A case study

Nicholson *et al.* presentan un caso de estudio donde se usa el lenguaje visual *Codecharts* para representar patrones de diseño. Utilizan herramientas de soporte desarrolladas en lenguaje Java para verificar el diseño, reportando las decisiones de diseño mal formadas.

El lenguaje *Codecharts* no se aplica directamente en el diseño. En realidad es usado en la codificación debido a que la herramienta aplicada verifica el código, lo compara con las definiciones de patrones de diseño creadas previamente en *Codecharts* y envía mensajes "pasa no pasa". Además de que está limitado para analizar código escrito únicamente en lenguaje de programación Java.

Comparado con el desarrollo que se presenta en esta investigación, se optó por trabajar con UML para que sea aplicado directamente en la etapa de diseño, sin limitarse a un lenguaje de programación en particular.

2.2 Design Pattern Support System: Help Making Decision in the Choice of Appropriate Pattern

Moudam *et.al.* presentan una herramienta llamada *Design Patterns Management System (DPMS)* que usa XML y XMI para modelar los patrones de diseño. Es usada para la categorización, administración e interrogación de una base de conocimientos, que puede ser extendida añadiendo nuevos patrones, usando principalmente los 23 patrones de diseño de Gamma *et al.* (Gamma, 2003). Esta herramienta está desarrollada en el lenguaje Java y es capaz de generar código en lenguaje Java y diagramas de clases en UML. Representan cada patrón de diseño en el lenguaje XML para evitar que los patrones sean repetidos. Cada patrón de diseño es representado con las siguientes características para hacerlo único:

- Intención
- Aplicabilidad
- Consecuencias
- Problemática
- Solución
- Alias
- Patrones relacionados
- Referencias

La herramienta usa XMI que presenta todas las características de la estructura de clases orientada a objetos lo cual permite generar el código correspondiente. Muestra una serie de situaciones que pueden ocurrir y el usuario selecciona una situación en particular y luego pulsa en "*Sugested Pattern*" para mostrar los patrones de diseño sugeridos.

Los autores aseguran que esta herramienta ayuda al usuario a seleccionar el patrón de diseño requerido para resolver un problema específico. También crean un administrador de palabras clave, para agregar nuevas palabras claves y nuevas situaciones a la base de conocimiento para los patrones de diseño, las nuevas situaciones se refieren a nuevos casos en el que patrón de diseño puede ser utilizado de forma correcta. El usuario requiere conocimiento para agregar nuevos patrones y las nuevas situaciones en estos patrones. Además debe leer las características de cada patrón sugerido para saber las consecuencias que puede tener el patrón.

En contraste con este trabajo de tesis, la herramienta de estos autores no considera la combinación de patrones, y está limitada a un único patrón por situación. El diagrama DPMS que se genera no puede ser modificado.

2.3 Recommendation system for design patterns in software development: An DPR overview

Palma *et.al.* describen la herramienta denominada *Design Pattern Recommender (DPR)*, la cual puede recomendar Patrones de diseño basado en *Goal-Question-Metric (GQM).*

El prototipo tiene dos niveles de solución. En el primer nivel DPR sólo propone uno o más patrones de diseño para resolver el contexto del problema. En el segundo nivel DPR propone un conjunto de patrones de diseño que combinados podrían resolver el problema, sin embargo no proporciona facilidades para realizar las combinaciones sugeridas. Se hacen una serie de preguntas al usuario. A cada respuesta se le asigna un "peso". Dependiendo del peso total obtenido se sigue un complejo diagrama realizado con anterioridad para sugerir el o los patrones de diseño.

El prototipo representa los patrones en XML e incluye los patrones *Adapter*, *Visitor* y *Decorator*. Las preguntas sólo apuntan a soluciones con estos patrones de diseño, por lo tanto, una desventaja es que con tan solo 3 patrones, el diagrama que se sigue es muy complejo, ampliarlo a los 23 patrones haría demasiado complejo al sistema, se tendrían que recalcular los pesos y las preguntas por cada nuevo patrón que se fuera a agregar.

2.4 A Recommendation System to Support Design Patterns Selection

Navarro *et.al.* presentan un módulo que recomienda patrones y está pensando como ampliación a la herramienta *Visually Enhanced and Interactive for Softgoal Interdependency Graphs (VEISIG).*

El modulo está enfocado para apoyar al usuario en diseños de arquitectura correctos, guiando al usuario a través de metas de diseño de forma visual. Utiliza un algoritmo de recomendación con el que se obtiene la "utilidad" (se refiere a qué tan útil sería para resolver el problema) de cada patrón para una solución concreta. Para la recomendación, utiliza un sistema de puntajes sin embargo, no ayuda al diseñador en su estructura, sólo recomienda los patrones a incluir en el diseño. En el estado actual de desarrollo del módulo no hay restricciones en cuanto a la recomendación de patrones que no deberían ser incluidos en la sugerencia, ya que sus consecuencias causan conflictos entre sí. El módulo no genera diagramas, ni considera la combinación de patrones.

2.5 Understanding the relevance of micro-structures for design patterns detection

Fontana *et.al.* proponen un método para detectar patrones de diseño en el código de sistemas en funcionamiento al buscar las micro-estructuras genéricas de cada uno. Estas micro-estructuras en común dan pistas de como los patrones de diseño fueron combinados.

La investigación se basa en buscar "pistas" de cada patrón de diseño en el código, como la declaración de clases; la información de múltiples clases; las variables; información de instancia; información de los métodos; información de retorno e información del lenguaje java. Del código analizado busca elementos que los autores categorizan como micro-estructura, los autores aseguran que cada patrón de diseño tiene un conjunto de micro-estructuras, definen qué estructuras son relevantes para *Singleton, Abstract Factory, Template Method, State, Composite y Decorator.* Recolectan la información y la comparan con las micro-estructuras de los patrones investigados para comparar qué patrones fueron usados en el código.

La investigación se basa principalmente en el análisis del código en proyectos o sistemas implementados para poder identificar patrones de diseño usados en él.

2.6 The Formal Research and Application Based on Design Patterns

Xiang *et.al.* proponen un lenguaje descriptivo orientado a objetos llamado *refinement Calculus of Object Systems(rCOS)* que incluye subclases; tipos de referencia; visibilidad; enlace dinámico; herencia; polimorfismo y declaraciones de variables locales anidadas; que permite definir formalmente. La sintaxis puede definir sistemas orientados a objetos, declaraciones de clases, comandos y expresiones en una gramática similar a Java. Los patrones de diseño en diagramas de clases y diagrama de secuencias definidos en UML.

La investigación presentada en el artículo define un conjunto de reglas formales para que pueda generar los diagramas de clases y de secuencias de un código especificado. Requiere aprender la definición de los patrones en este nuevo lenguaje. Sólo genera el diagrama de clases y el diagrama de secuencia basado en las especificaciones dadas por el usuario. No considera la combinación de patrones en su generación y sólo genera un patrón a la vez.

2.7 Checking UML Design Patterns in Java Implementations

Pires *et.al.* publicaron en un artículo anterior un método para verificar la implementación en Java con su modelo UML equivalente en UML. En este artículo, proponen un método para aplicar patrones de diseño basándose en perfiles de UML. Los autores crean un conjunto de perfiles en UML que incorporan los patrones de diseño en los diagramas de clase de UML, un conjunto de pruebas de diseño para verificar que el patrón de diseño se implemente de forma correcta en el código de Java y una arquitectura dirigida por modelos para aplicar estas pruebas de forma automática.

Los autores crean un conjunto de pruebas para verificar si cada patrón de diseño cumple el perfil de UML correspondiente al que fue definido. El perfil de UML está basado en estereotipos adicionales y valores de etiquetas que son aplicados a los elementos de UML, como clases, métodos, atributos, asociaciones, etc. Los perfiles que puede verificar son de los patrones de diseño *Facade, Singleton, Proxy, Builder, Abstract Factory, Factory Method y Prototype*. Los autores mencionan una herramienta que desarrollaron, la cual incorpora todos los conceptos mencionados y verifica el código existente para comprobar si los patrones de diseño fueron implementados de forma correcta.

La herramienta requiere desarrollar un conjunto de perfiles para cada patrón de diseño, cumpliendo los estereotipos para buscar patrones en sistemas implementados.

En contraste con la tesis a desarrollar, esta investigación no considera la combinación de patrones de diseño.

2.8 A Visual Language for Design Pattern Modelling and Instantiation

Maplesden *et.al*. presentan un **Lenguaje de modelado de patrones de diseño (DPML)**. Este es un lenguaje visual para modelar patrones de diseño y muestran cómo puede ser instalado en lenguajes orientados a objetos. Usando este lenguaje desarrollaron dos herramientas prototipo, *DPTool* y *MaramaDPTool*, las cuales se integran al ambiente de programación *Eclipse* permitiéndo generar código.

Este lenguaje define un metamodelo y una notación para la solución de cada uno de los patrones de diseño e instancias de solución de los objetos modelados. Al modelar con DPML, las especificaciones de cada patrón pueden ser transformadas en UML.

Los autores refieren que el modelado de patrones se vuelve complicado y es necesario ampliar la especificación de cada patrón. La forma de representación de patrones de diseño es muy compleja lo que complica la generación de código. No considera la combinación de patrones de diseño.

2.9 Automatic verification of design patterns in Java

Blewitt *et.al* presentan un lenguaje para la especificación de patrones de diseño llamado *SPINE* basado en lenguaje PROLOG el cual permite especificar patrones en términos de funciones y predicados para verificar que estén definidos de forma correcta. Desarrollaron una herramienta llamada *HEDGEHOG* que analiza un código fuente en el lenguaje Java que utiliza las restricciones de *SPINE* para verificar que los patrones estén formados de acuerdo a las especificaciones previamente definidas y lo compara con el código fuente en Java para detectar si la clase cumple con el patrón de diseño seleccionado.

Considera la combinación de patrones uniendo dos reglas con “and” y “or” como una expresión lógica, lo cual sólo verifica que ambos patrones cumplan sus respectivas reglas pero no verifica ninguna unión estructural.

Los autores especifican que se pueden construir la mayoría de los patrones de diseño del catálogo de Gamma, sin embargo los patrones de diseño *Builder, Facade, Chain of Responsability, Command, Interpreter, Mediator y Mememnto* **NO** pueden ser representados en este lenguaje *SPINE* porque al definirlos son demasiado abstractos pueden ser muy restringidos (lo que produce muchos falsos negativos) o muy vagos (lo que produce muchos falsos positivos) , por lo tanto es imposible verificarlos usando la herramienta *HEDGEHOG*.

2.10 pLinker: Relaciones con Patrones de diseño

Liebener *et.al* proponen una herramienta que es capaz de hacer diagramas de patrones de diseño UML y agregar patrones de diseño en la estructura. La herramienta desarrollada pLinker es un editor gráfico de clases UML el cual incorpora facilidades para crear patrones de diseño, esta herramienta permite crear tu propio catálogo de patrones de diseño y relacionarlos entre sí. La herramienta tiene dos modos de trabajo:

El primero permite al usuario modelar diseños de arquitecturas de clases, interfaces y relaciones entre las mismas (herencia, uso e implementación). En este modo los autores prometen que la herramienta puede detectar qué patrón de diseño fue utilizado, aseguran que es posible agregar patrones de diseño a la estructura ya creada.

El segundo modo es la gestión del catálogo de patrones donde el usuario agrega patrones de diseño llenando una plantilla con los datos que son Nombre, Consecuencias, Clases participantes. Debe indicar qué tipo de relación tendrá con otro patrón de diseño, qué clases se relacionan o qué clase será remplazada, en qué forma estructural lo harán (por ejemplo: la Clase A y la Clase B es la misma en ambos patrones de diseño, se unen estructuralmente).

El usuario es capaz de instanciar un patrón de diseño basándose en las especificaciones dadas por el usuario crea la estructura del patrón de diseño deseado. Debe agregar clases al "diagramador" luego puede seleccionar una instancia de un patrón de diseño que se aplique con esas clases. La herramienta otorga la estructura del patrón de diseño instanciado y muestra una serie de los posibles patrones que pueden ser combinados con cada patrón. Al combinar los patrones vuelve a cambiar la forma estructural de acuerdo al patrón seleccionado y acorde a la plantilla que fue llenada. La herramienta muestra con qué patrón de diseño se puede relacionar un patrón actual.

Los autores definen las combinaciones de patrones de diseño usando un documento de tipo plantilla que rellenan para cada combinación entre patrones. En esta plantilla marcan el tipo de relación que tiene un patrón de diseño con otros patrones de diseño, las consecuencias de aplicar ambos, el momento para aplicarlo, los cambios que se harán a la estructura del patrón original y qué acciones deben ser tomadas después de aplicar la combinación.

La herramienta limita las relaciones sólo a los patrones pre-establecidos y sólo muestra *Composite e Iterator*, y las combinaciones únicamente se pueden hacer en los mismos patrones pre-establecidos.

Esta herramienta necesita que el usuario sea un experto en patrones de diseños para poder llenar la plantilla de forma correcta y evitar malos diseños.

2.11 Integration of Design Patterns into Object-Oriented Design using Rational Rose

Mannhaput propone un prototipo para aplicar patrones de diseño en la herramienta Rational Rose menciona como desarrollar los patrones de diseño en este prototipo. Siguiendo un "modelo de patrón", que identifica los componentes (clases, sus propiedad y atributos), atributos (internos o externos del patrón de diseño) y métodos (internos o externos, describe el comportamiento del patrón de diseño).

Este autor menciona que los patrones de diseño pueden ser "juntados como mosaicos o tejas", este enfoque aclama que dos patrones de diseño pueden ser juntados para generar un tercero, por ejemplo el patrón *Interpreter* , *Visitor* y otros patrones pueden ser juntados para generar un mosaico que contenga el patrón *Visitor* e *Interpreter* por medio de la clase "client".

El prototipo permite al usuario generar un nuevo elemento en el diagrama llamado "NewPattern", el usuario puede "entrar" (haciendo doble clic en él) y diagramar los elementos del patrón de diseño. Puede realizar combinaciones usando una ventana de diálogo, en la cual el usuario puede seleccionar los patrones a combinar, qué elementos mezclar, y qué componentes, atributos y operaciones renombrar. En la ventana mostrada el usuario selecciona dos componentes, y luego pulsa el botón "Merge" lo cual une ambos patrones con los componentes señalados.

El diagrama de clases resultante es un conjunto de clases relacionadas, y los elementos "Patrones de diseño" se muestran como cajas grises con el nombre del patrón de diseño.

Esta forma de combinar no sigue ninguna regla de combinación por lo tanto la herramienta no puede verificar la correcta combinación de patrones de diseño y el usuario podría combinar los patrones de una forma incorrecta, sin percatarse de ello.

2.12 Comparativa de estudios

Constantemente la literatura menciona que es muy importante una definición formal de los patrones de diseño para evitar los malos diseños.

Con respecto a la combinación de patrones, muchos diseñadores sólo suponen que "simplemente se hace" pero una mala combinación puede llevar a defectos y empeorar el tiempo de desarrollo.

A continuación se muestra una tabla comparativa de los 6 trabajos relacionados estudiados, que más se apegan a nuestro objetivo. La comparativa muestra el nombre de la herramienta, la referencia del trabajo e incluye los patrones de diseño que son usados para las pruebas.

Herramienta	Patrones usados	Realiza combinación	Etapa aplicada	Usa UML	Entrada	Salida
(Pires, 2010)	*Facade,Singleton, Proxy, Builder, Abstract Factory, Factory Method, Prototype*	No	Codificación	Si	Código en Java	Avisa si el patrón fue creado de forma correcta
DPML (Maplesden, 2007)	*Abstract Factory*	No	Diseño	Si	Diagramas hechos por el usuario	Diagramas de clase y modelos DPML
SPINE (Blewitt, 2005)	*Todos* (Gamma, 2003) *excepto Builder, Facade, Chain of Responsability, Command,Interpreter, Mediator y Memento.*	No estructuralmente	Codificación	No	Código en Java	Avisa si el patrón cumple las reglas especificadas
pLinker (Liebener, 2003)	*Composite, Iterator*	Si	Diseño	Si	Diagramas hechos por el usuario	Diagrama de clases
Rational Rose (Mannhaput, 2000)	*Iterator, Composite, Factory Method*	Si	Diseño	Si	Diagramas hechos por el usuario	Diagramas de clase
Lo que propone este trabajo	*Composite, Strategy, Iterator, Factory Method, Template Method, Observer*	Si	Diseño	Si	Diagramas hechos por el usuario	Diagramas de clases y avisos si el patrón o la combinación se hacen de acuerdo a las reglas

Tabla 1. Tabla comparativa de estudios

CAPÍTULO 3: MARCO TEÓRICO

3.1 Análisis de combinación de patrones de diseño

En este capítulo se describe la investigación realizada sobre la teoría de las gramáticas relacionales usada en la tesis de Sergio González (Gonzáles, 2010) para definir los patrones de diseño *Iterator, Observer y Composite*. También fueron usadas por Héctor Uriel Pérez Rojas (Pérez, 2014) para definir los patrones de diseño *Strategy, Composite, Templated Method y Factory Method*. También se analizó la combinación de los patrones de diseño que fueron agregados a SiVerPat los cuales son:

1. ***Strategy-Composite***
2. ***Strategy-Template Method***
3. ***Strategy-Factory Method***
4. ***Composite- Template Method***
5. ***Composite-Factory Method***
6. ***Template Method – Factory Method***

Los patrones de diseño no se pueden combinar con ellos mismos (por ejemplo *Strategy-Strategy*). Cada patrón tiene diferentes formas de ser representado pero sólo se tomó en cuenta la estructura básica.

En (Pérez, 2014) y (Gonzáles, 2010) se declararon dos tipos de regla para cada patrón: una forma estática (con el mínimo número de elementos) y una forma dinámica (con una o más instancias de la clase). Ya que el patrón dinámico contiene al patrón estático, la regla a capturar será la del patrón dinámico.

Se hicieron los diagramas de los patrones de diseño y sus combinaciones en idioma inglés. A continuación una descripción de cada patrón, sus combinaciones y los roles que realiza cada clase dentro de cada patrón.

La combinación de patrones de diseño se realiza estructuralmente, es decir modificando la estructura del diagrama de clases, existen patrones de diseño (por ejemplo, Template Method, Factory Method, etc) que describen un comportamiento (por ejemplo, derivar métodos a otra clase) especificar este comportamiento requiere un contexto por lo tanto SiVerPat omite el comportamiento y se enfoca en la estructura.

3.2. Gramáticas Relacionales

Las gramáticas relacionales son un formalismo sintáctico para describir lenguajes visuales, en esta gramática cada sentencia en un lenguaje es representado como un conjunto de objetos visuales y un conjunto de relaciones entre esos objetos (Ferrucci, 1996). La diferencia entre las gramáticas visuales es que las gramáticas relacionales se enfocan en el origen, destino y tipo de las relaciones entre los elementos, mientras que las visuales en la posición de la que salen o entran. Estas gramáticas son denominadas gramáticas posicionales.

3.2.1 Gramática Símbolo-Relación

Las gramáticas Símbolo-Relación es la evolución de las gramáticas relacionales, son un modelo de gramáticas de relaciones capaz de describir cualquier lenguaje gráfico.

En la gramática Símbolo-Relación, cada enunciado en un lenguaje es representado como un conjunto de objetos visuales (s-items) y un conjunto de relaciones entre estos objetos (r-items)

3.2.2 Cadena Símbolo-Relación

Dado un alfabeto de símbolos terminales $\mathbf{V_T}$ y un conjunto de símbolos de relación $\mathbf{V_R}$, una cadena símbolo-relación *w* en $\mathbf{V_T}$ y $\mathbf{V_R}$ es un par **<M,R>,** donde:

M: conjunto de s-items (v, i) con $v \in V_T$ e $i \in N$, el conjunto de números naturales. Por simplicidad, cada s-item (v, i) en M será escrito como v^i, en donde el superíndice es usado para distinguir diferentes repeticiones del mismo símbolo.

R: conjunto de r-items de la forma $r(t_1, t_2)$con $t_1, t_2 \in M$ y $r \in V_R$ indicando que se mantiene una relación entre t_1 y t_2 .

3.2.3 Definición formal de Gramática símbolo-relación

Una gramática símbolo-relación es una séxtupla $G = (V_N, V_T, V_R, S, P, R)$ donde:
V_N = Conjunto de símbolos no terminales.
V_T = Conjunto de símbolos terminales.
V_R = Conjunto de símbolos de relación.
S = Símbolo inicial.
P = Conjunto finito de producciones s-items:

$Y^0 \rightarrow$ <M, R>

Dónde:

a) <M, R> es enunciado en V_R y $V_N \cup V_T$
b) $Y \in V_N$

R= Conjunto finito de r-producciones, de la forma:

$s(Y^0, X^1) \rightarrow [I]Q$ ó
$s(X^1, Y^0) \rightarrow [I]Q$

Dónde:

c) I es la etiqueta de una s-producción $I: Y^0 \rightarrow < M, R >$ (la s-producción después de la cual se debe aplicar la r-producción),
d) $X \in V_N \cup V_T$ (donde X puede ser un símbolo terminal o no terminal)
e) $Q \neq \emptyset$ es un conjunto de r-items de la forma $r(Z, X^1)$ ó $r(X^1, Z)$, con $Z \in M$ (Cuando existe un elemento Q, éste debe ser un conjunto de r-items donde se describa el tipo de relación "r", Z pertenece al conjunto de símbolos terminales y no terminales, y X permanece como se describe en su forma original.) (Ferrucci, 1996)

3.2.4 Ejemplo

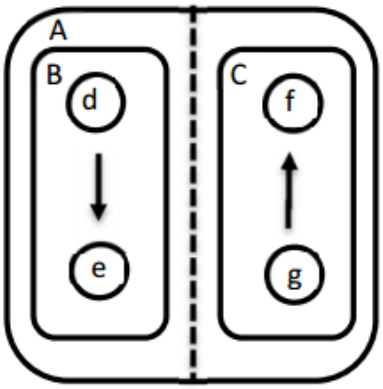

Figura 1.Ejemplo de gramática Símbolo - Relación

La figura anterior puede ser representada por siguiente sentencia.

w = <M, R> en $\mathbf{V_T}$ = {caja, círculo, barra, flecha} V_R = {contiene, izquierda, inicia, termina}, donde:

M = {caja^1, caja^2, caja^3, barra^1, flecha^1, flecha^2, circulo^1, circulo^2, circulo^3, circulo^4}

R = {contiene(caja^1, caja^2), contiene(caja^1, caja^3), contiene(caja^1, barra^1), contiene(caja^2, circulo^1), contiene(caja^2, circulo^2), contiene(caja^3, circulo^3), contiene(caja^3, circulo^4), inicia(flecha^1, circulo^1), termina(flecha^1, circulo^2), inicia(flecha^2, circulo^4), termina(flecha^2, circulo^3), izquierda(caja^2, barra^1), izquierda(barra^1, caja^3)}

3.2.5 Definición de una clase

Sea *AT* el conjunto de atributos tal que:

$$AT = \{a | \forall a\ A(a)\}$$

Dónde:

A(a) = "a es atributo de una clase"

Sea *MT* el conjunto de métodos tal que:

$$MT = \{m\ | \forall m(M(m)\ \wedge (Mi(m)\ \vee\ Ma(m)))\}$$

Dónde:

M(m) = "m es un método"
Mi(m) = "m es un método implementado"
Ma(m) = "m es un método abstracto"

3.2.6 Definición de clases abstractas

Sea Ca es el conjunto de clases abstractas, definido de la siguiente manera:

$$Ca = \{x | \forall x\ C(x) \wedge Ma(x)\}$$

Cada uno de los elementos del conjunto Ca cumple con las siguientes propiedades:

- C(x)="x es una clase"
- Ma(x) = "x tiene al menos un método abstracto"

3.2.7 Definición de clases concretas

Sea Cc es el conjunto de clases abstractas, definido de la siguiente manera:

$$Cc = \{x | \forall x\ C(x) \wedge Mi(x) \wedge \neg Ma(x)\}$$

Cada uno de los elementos del conjunto Ca cumple con las siguientes propiedades:

- C(x)=”x es una clase”
- Ma(x) = “x tiene al menos un método abstracto”
- Mi(x) = “x tiene métodos implementados”

3.2.9 Nomenclatura utilizada en el transcurso de la tesis

En esta sección se usará la nomenclatura usada en el trabajo de (Gonzáles, 2010) y (Pérez, 2014), los elementos importantes son:

- Cada relación entre clases, sea cual sea el tipo de la relación, será interpretada en el sentido de la flecha, siendo el primer término de **r-item** la clase origen de la relación, mientras que el segundo término será la clase destino de la relación.

- El superíndice representa el identificador de la clase partiendo de ^{2} ya que el superíndice ^{1} representa un comodín que sustituye al elemento representado, y el superíndice ^{0} representa al elemento a sustituir dentro del **s-item**.

3.2.10 Diagrama de clases

Cada patrón de diseño cuenta con diagramas de clases diferentes

- Diagrama de clases estático: Se refiere al patrón de diseño con el número mínimo de elementos de la estructura del patrón.

- Diagrama de clases dinámico: Se refiere a que el patrón de diseño puede existir una o más instancias de la clase que puede tener algoritmos diferentes.

3.2.11 Herencia

Este tipo de relación representa que una clase padre hereda tanto atributos como comportamiento a las clases hijas heredadas. La representación gráfica que se usa es una flecha con punta en triángulo vacío o blanco. La clase heredada será la clase origen de la relación, mientras que la clase que hereda será el destino de la relación.

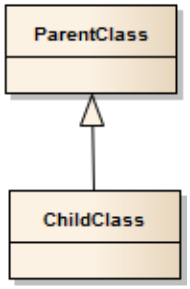

Figura 2. Representación de la herencia

En la gramática símbolo-relación, es descrita como:

Her(ClasePadre^2, ClaseHija^2)

Al cambiarla al idioma inglés:

Inh(ParentClass^2, ChildClass^2)

3.2.12 Agregación fuerte o Composición

Se representa por una línea en un extremo (destino), y un diamante relleno en su lado opuesto (origen). Este tipo de relación trata sobre un objeto de una clase que contiene a uno o varios objetos agregados de otra o de la misma clase.

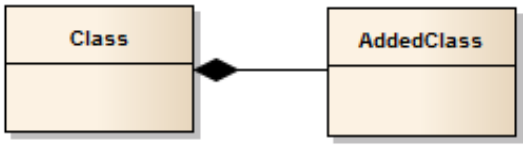

Figura 3. Representación de agregación fuerte

En la gramática símbolo-relación, el símbolo de la relación para una Agregación Fuerte será describe como:

AggF(Clase^2, ClaseAgregada^2)

La gramática en inglés:

Comp(Class^2, AddedClass^2)

3.2.13 Agregación débil o Agregación

Se representa por una línea en un extremo (destino), y un diamante vacío en su otro extremo (origen). Este tipo de relación se trata sobre un objeto de una clase que contiene una referencia a objetos de otra o de la misma clase, siendo esta última la clase agregada.

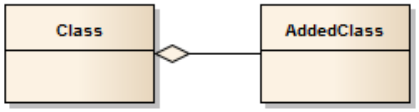

Figura 4. Representación de agregación débil

En la gramática símbolo-relación, el símbolo de la relación para una Agregación Débil se describe como:

AggD(Clase^2, ClaseAgregada^2)

La gramática en inglés:

Agg(Class^2, AddedClass^2)

3.2.14 Instanciación

Se representa por una línea punteada con una flecha en un extremo (destino) que representa la clase instanciada, mientras en el otro extremo (origen) se representa la clase que instancia.

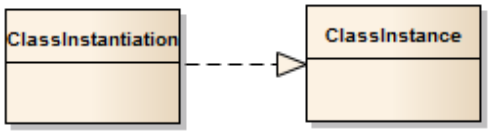

Figura 5. Representación de instanciación

En la gramática símbolo-relación, el símbolo de la relación para una instanciación se describe como:

Ins(ClaseInstanciadora^2, ClaseInstanciada^2)

La gramática en inglés:

Ins(ClassInstantiation^2, ClassInstance^2)

3.3 Patrón Strategy

Es parte de los patrones de comportamiento. La intención de este patrón de diseño es la elección en tiempo de ejecución de un comportamiento estratégico que sea el más conveniente para la solución del problema.

3.3.1 Diagrama de clases estático

El diagrama de clases del patrón de diseño ***Strategy*** en su forma estática es el siguiente:

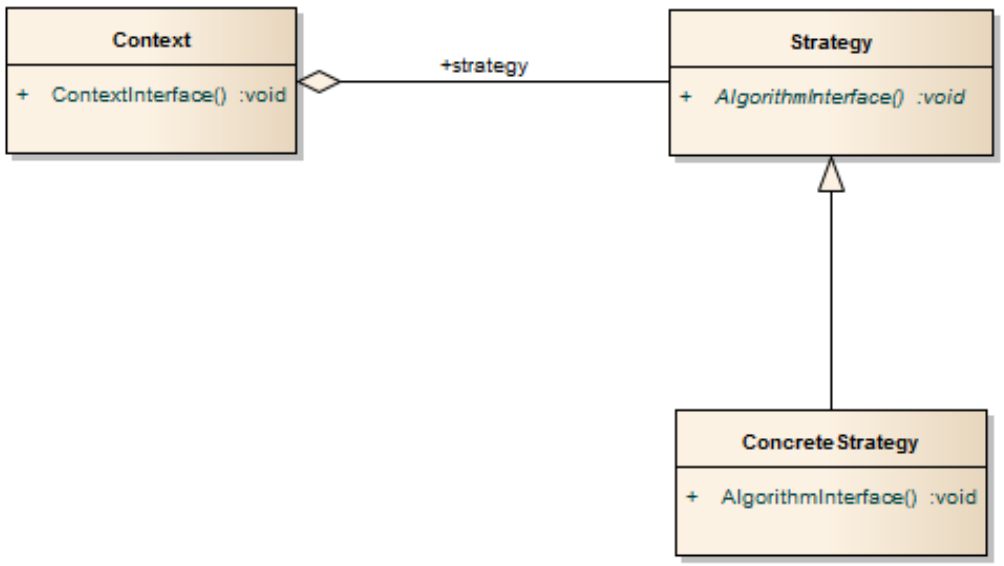

Figura 6. Patrón de diseño Strategy en su representación Estática (Pérez, 2014)

3.3.2 Diagrama de clases dinámico

El diagrama de clases del patrón de diseño *Strategy* en su forma dinámica es el siguiente:

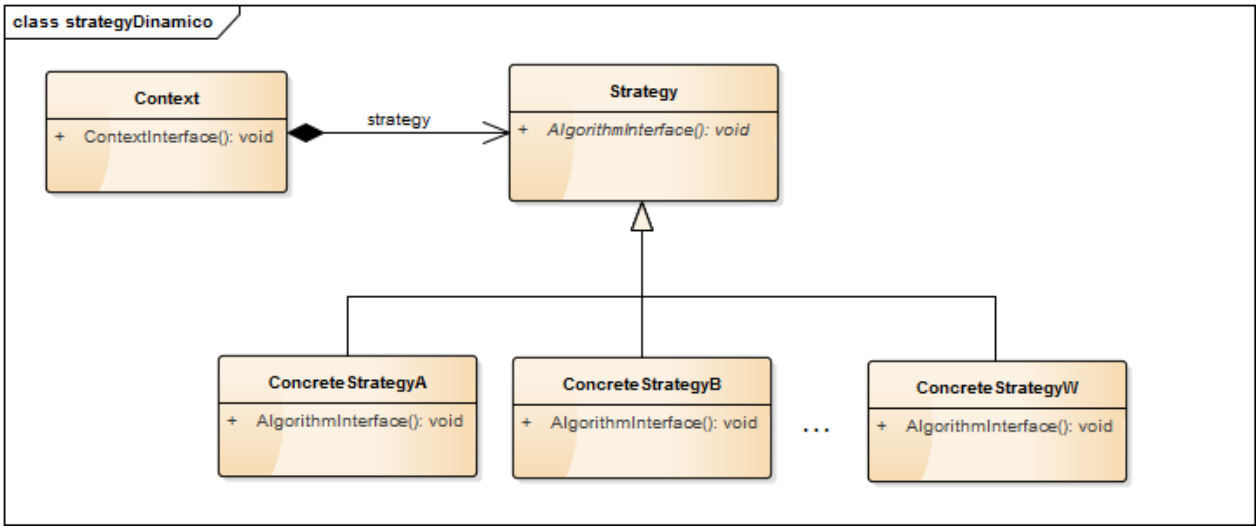

Figura 7. Patrón de diseño Strategy en su representación Dinámica (Pérez, 2014)

3.3.3 Roles del patrón Strategy

El patrón ***Strategy*** define 3 roles diferentes: *Context, Strategy y ConcreteStrategy*. Dentro de esta estructura, podemos encontrar una o más clases que ocupen el rol de *ConcreteStrategy* (ConcreteStrategyA, ConcreteStrategyB, ConcreteStrategyC… ConcreteStrategyW). (Pérez, 2014)

3.3.4 Definición de la regla gramatical del patrón de diseño Strategy estático

StrategyPattern = (
{Z, CX, ST, CST},
{context, strategy, concreteStrategy},
{Inh, Comp},
Z,
P,
R
)

P:

0: $Z^0 \rightarrow < \{CX^2, ST^2, CST^2\}, \{Inh(CST^2, ST^2), Comp(CX^2, ST^2)\} >$
1: $CX^0 \rightarrow < \{context^2\},\ \emptyset >$
2: $ST^0 \rightarrow < \{strategy^2\},\ \emptyset >$
3: $CST^0 \rightarrow < \{concreteStrategy^2\},\ \emptyset >$

R:

1: $Comp(CX^0, \tau^1) \rightarrow [1]\{Comp(context^2, \tau^1)\}$
$\tau: ST, strategy$
2: $Comp(\sigma^1, ST^0) \rightarrow [2]\{Comp(\sigma^1, strategy^2)\}$
$\sigma: CX, context$
3: $Inh(\tau^1, ST^0) \rightarrow [2]\{Inh(\tau^1, strategy^2)\}$
$\tau: CST, concreteStrategy$
4: $Inh(CST^0, \sigma^1) \rightarrow [1]\{Inh(concreteStrategy^2, \sigma^1)\}$
$\sigma: ST, strategy$

3.3.5 Definición de la regla gramatical del patrón de diseño Strategy dinámico

StrategyPattern = (
{Z, CX, ST, CST, CSST, CSSTX},
{context, strategy, concreteStrategy},
{Inh, Comp},
Z,
P,
R
)

P:

0: $Z^0 \rightarrow < \{CX^2, ST^2, CSST^2\}, \{Inh(CSST^2, ST^2), Comp(CX^2, ST^2)\} >$
1: $CSST^0 \rightarrow < \{CST^2, CSSTX^2\},\ \emptyset >$
2: $CSSTX^0 \rightarrow < \{CSST^2\},\ \emptyset >$
3: $CSSTX^0 \rightarrow < \{\lambda\}, \emptyset >$
4: $CST^0 \rightarrow < \{concreteStrategy^2\},\ \emptyset >$
5: $ST^0 \rightarrow < \{strategy^2\},\ \emptyset >$
6: $CX^0 \rightarrow < \{context^2\},\ \emptyset >$

R:

0: Inh$(CSST^0, \tau^1) \rightarrow [1]\{Inh(CST^2, \tau^1), Inh(CSSTX^2, \tau^1)\}$
$\tau: ST, Strategy$
1: Inh$(CSSTX^0, \sigma^1) \rightarrow [2]\{Inh(CSST^2, \sigma^1)\}$
$\sigma: ST, strategy$
2: Inh$(CSSTX^0, \tau^1) \rightarrow [3]\emptyset$
$\tau: ST, Strategy$
3: Inh$(CST^0, \sigma^1) \rightarrow [4]\{Inh(concreteStrategy^2, \sigma^1)\}$
$\sigma: ST, strategy$
4: $Comp(\tau^1, ST^0) \rightarrow [5]\{Comp(\tau^1, strategy^2)\}$
$\tau: CX, context$
5: $Comp(CX^0, \sigma^1) \rightarrow [6]\{Comp(context^2, \sigma^1)\}$
$\sigma: ST, strategy$

3.4. Patrón Template Method

Forma parte de los patrones de comportamiento. La intención de este patrón de diseño se enfoca en la implementación de un método en la clase padre que lleva la orquestación del proceso de ejecución de los métodos definidos tanto en la clase base como en las clases derivadas. El comportamiento final varía dependiendo del comportamiento definido en las clases derivadas.

3.4.1 Diagrama de clases estático

El diagrama de clases del patrón de diseño *Template Method* en su forma estática es el siguiente:

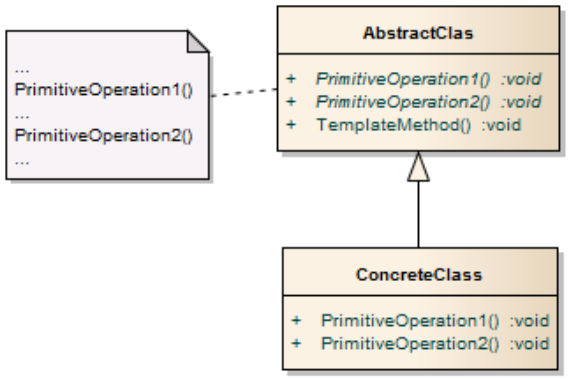

Figura 8. Patrón de diseño Template Method en su representación Estática (Pérez, 2014)

3.4.1 Diagrama de clases dinámico

El diagrama de clases del patrón de diseño *Template Method* en su forma dinámica, es decir, que existen una o más instancias de la clase *ConcreteClass* es el siguiente:

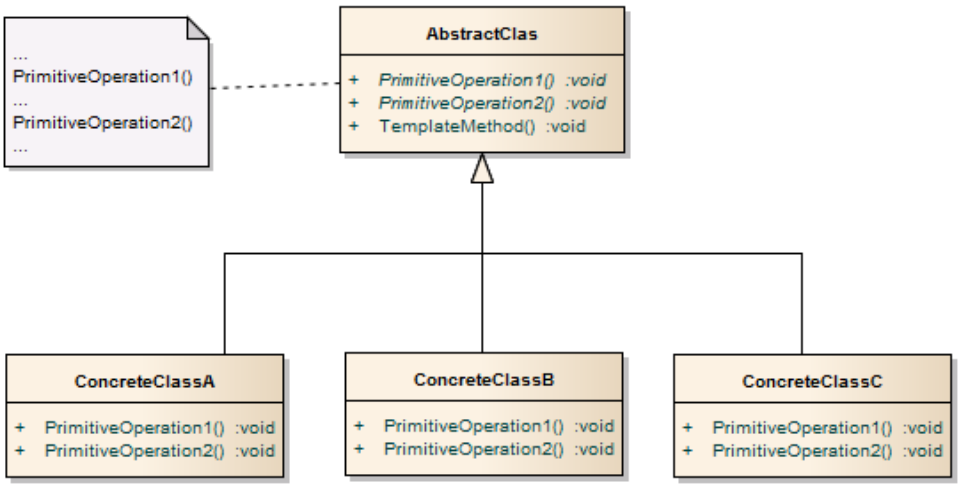

Figura 9. Patrón de diseño Template Method en su representación Dinámica (Pérez, 2014)

3.4.2 Roles del patrón Template Method

El patrón ***Template Method*** define 2 roles diferentes: *AbstractClass* y *ConcreteClass*. Dentro de esta estructura, podemos encontrar una o más clases que ocupen el rol de *ConcreteClass*.

3.4.3 Definición de la regla gramatical del patrón de diseño Template Method estático

TemplateMethodPattern = (
{Z, AC, CC},
{abstractClass, concreteClass},
{Inh},
Z,
P,
R
)

P:

0: $Z^0 \rightarrow < \{AC^2, CC^2\}, \{Inh(CC^2, AC^2)\} >$
1: $AC^0 \rightarrow < \{abstractClass^2 = \{x | \forall x(C(x) \wedge M(x) \wedge Ma(x) \wedge Mt(x))\}\}, \emptyset >$
2: $CC^0 \rightarrow < \{concreteClass^2 = \{x | \forall x(C(x) \wedge Mi(x) \wedge \neg Ma(x)\ \}\}, \emptyset >$

Dónde:
C(x) = “x es una clase”
Mi(x) = “x tiene métodos implementados”
Ma(x) = “x tiene al menos un método abstracto”
M(x) = “x tiene métodos”
Mt(x) = “x tiene un método TemplateMethod”

R:

1: Inh(τ^1, AC^0) → [1]$\{Inh(\tau^1, abstractClass^2)\}$
$\tau: CC, concreteClass$
2: Inh(CC^0, σ^1) → [2]$\{Inh(concreteClass^2, \sigma^1)\}$
$\sigma: AC, abstractClass$

3.4.4 Definición de la regla gramatical del patrón de diseño Template Method dinámico

TemplateMethodPattern = (
{Z, AC, CCD, CCX, CC},
{abstractClass, concreteClass},
{Inh},
Z,
P,
R
)

P:

0: $Z^0 \rightarrow < \{AC^2, CCD^2\}, \{Inh(CC^2, AC^2)\} >$
1: $CCD^0 \rightarrow < \{CC^2, CCX^2\}, \emptyset >$
2: $CCX^0 \rightarrow < \{CCD^2\}, \emptyset >$
3: $CCX^0 \rightarrow < \{\lambda\}, \emptyset >$
4: $AC^0 \rightarrow < \{abstractClass^2 = \{x | \forall x(C(x) \wedge M(x) \wedge Ma(x) \wedge Mt(x))\}\}, \emptyset >$
5: $CC^0 \rightarrow < \{concreteClass^2 = \{x | \forall x(C(x) \wedge Mi(x) \wedge \neg Ma(x)\}\}, \emptyset >$

Donde:
C(x) = “x es una clase”
Mi(x) = “x tiene métodos implementados”
Ma(x) = “x tiene al menos un método abstracto”
M(x) = “x tiene métodos”
Mt(x) = “x tiene un método TemplateMethod”

R:

0: Inh(CCD^0, τ^1) → [1]$\{Inh(CC^2, \tau^1), Inh(CCX^2, \tau^1)\}$
$\tau^1: AC, abstractClass$
1: Inh(CCX^0, σ^1) → [2]$\{Inh(CCD^2, \sigma^1)\}$
$\sigma^1: AC, abstractClass$
2: Inh(CCX^0, τ^1) → [3] $\emptyset$
$\tau^1: AC, abstractClass$
3: Inh(σ^1, AC^0) → [4]$\{Inh(\sigma^1, abstractClass^2)\}$
$\sigma^1: CC, concreteClass$
4: Inh(CC^0, τ^1) → [2]$\{Inh(concreteClass^2, \tau^1)\}$
$\tau^1: AC, abstractClass$

3.5 Patrón Factory Method

Forma parte de los patrones creacionales. La intención de este patrón de diseño se enfoca en definir una interfaz para la creación de objetos, permitiendo a las subclases decidir qué clase instanciar.

3.5.1 Diagrama de clases estático

El diagrama de clases del patrón de diseño *Factory Method* en su forma estática es el siguiente:

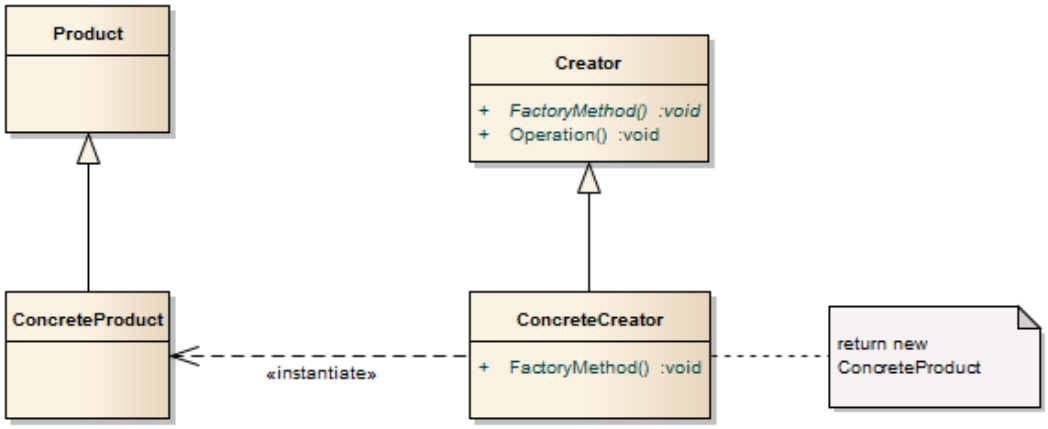

Figura 10. Patrón de diseño Factory Method en su representación Estática (Pérez, 2014)

3.5.2 Diagrama de clases dinámico

El diagrama de clases del patrón de diseño *Factory Method* en su forma dinámica, es decir, que existen una o más instancias de la clase *ConcreteProduct*, así como una o más instancias de la clase *ConcreteFactory* es el siguiente:

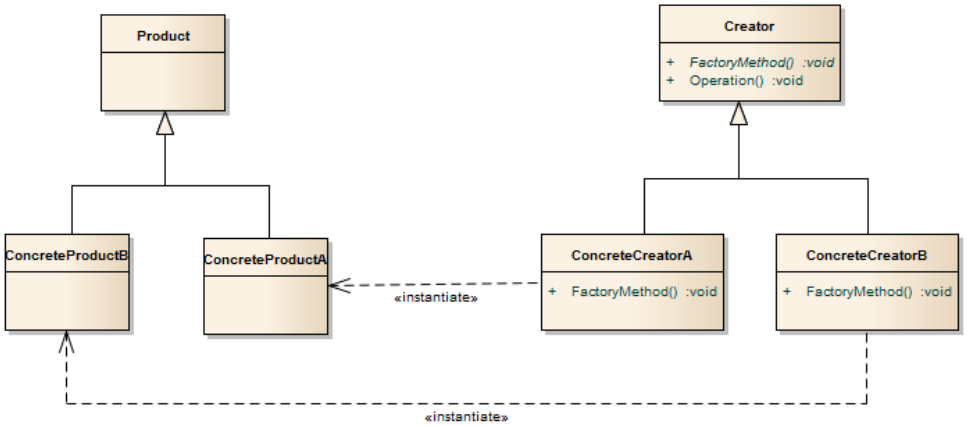

Figura 11. Patrón de diseño Factory Method en su representación Dinámica (Pérez, 2014)

3.5.3 Roles del patrón Factory Method

El patrón ***Factory Method*** define 4 roles diferentes: *Product*, *ConcreteProduct*, *Creator* y *ConcreteCreator*. Dentro de esta estructura, podemos encontrar una o más clases que ocupen el rol de *ConcreteProduct*, además, una o más clases que ocupen el rol *ConcreteCreator*.

3.5.4 Definición de la regla gramatical del patrón de diseño Factory Method estático

FactoryMethodPattern = (
{Z, CRT, CCRT, P, CP},
{creator, concreteCreator, product, concreteProduct},
{Inh, Ins},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow <\{CRT^2, CCRT^2, P^2, CP^2\}, \{Inh(CCRT^2, CRT^2), Inh(CP^2, P^2), Ins(CCRT^2, CP^2)\}>$
1: $CRT^0 \rightarrow < \{creator^2 = \{x | \forall x(C(x) \wedge M(x) \wedge Mfa(x))\}\}, \emptyset >$
2: $CCRT^0 \rightarrow < \{concreteCreator^2 = \{x | \forall x(C(x) \wedge Mfi(x) \wedge \neg Mfa(x)\}\}, \emptyset >$
3: $P^0 \rightarrow < \{product^2\}, \emptyset >$
4: $CP^0 \rightarrow < \{concreteProduct^2\}, \emptyset >$

Dónde:
C(x) = “x es una clase”
Mfi(x) = “x tiene método Factory Method implementado”
Mfa(x) = “x tiene un método Factory Method abstracto”
M(x) = “x tiene o no método”

R:
1: $Inh(\tau^1, CRT^0) \rightarrow [1]\{Inh(\tau^1, creator^2)\}$
$\tau: CCRT, concreteCreator$
2: $Inh(CCRT^0, \sigma^1) \rightarrow [2]\{Inh(concreteCreator^2, \sigma^1)\}$
$\sigma: CRT, creator$
3: $Ins(CCRT^0, \tau^1) \rightarrow [2]\{Ins(concreteCreator^2, \tau^1)\}$
$\tau: CP, concreteProduct$
4: $Inh(\sigma^1, P^0) \rightarrow [3]\{Inh(\sigma^1, product^2)\}$
$\tau: CP, concreteProduct$
5: $Inh(CP^0, \tau^1) \rightarrow [4]\{Inh(concreteProduct^2, \tau^1)\}$
$\tau: P, product$
6: $Ins(\sigma^1, CP^0) \rightarrow [4]\{Ins(\sigma^1, concreteProduct^2)\}$
$\tau: CCRT, concreteCreator$

3.5.5 Definición de la regla gramatical del patrón de diseño Factory Method dinámico

FactoryMethodPattern = (
{Z, CPS, CPSX, P, CRT, CP, CCRT},
{product, creator, concreteProduct, concreteCreator},
{Inh, Ins},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow <\{P^2, CRT^2, CPS^2\}, \{Inh(CPS^2, P^2), Inh(CPS^2, CRT^2)\}>$
1: $CPS^0 \rightarrow <\{CP^2, CPSX^2, CCRT^2\}, \{Ins(CCRT^2, CP^2)\}>$
2: $CPSX^0 \rightarrow <\{CPS^2\}>$
3: $CCX^0 \rightarrow < \{\lambda\}, \emptyset >$
4: $P^0 \rightarrow < \{product^2\}, \emptyset >$

5: CRT^0 → < $\{creator^2 = \{x|\forall x(C(x)\text{^}M(x)\text{^}\ Mfa(x))\}\},\ \emptyset$ >
6: CP^0 → < $\{concreteProduct^2\}, \emptyset$ >
7: CCRT^0 → < $\{concreteCreator^2 = \{x|\forall x(C(x)\ \text{^}\ Mfi(x)\ \ \text{^}\ \neg Mfa(x)\ \}\},\ \emptyset$ >

Dónde:

C(x) = “x es una clase”
Mfi(x) = “x tiene método Factory Method implementado”
Mfa(x) = “x tiene un método Factory Method abstracto”
M(x) = “x tiene o no método”

R:

0: Inh$(CPS^0, \tau^1)\ \rightarrow [1]\{Inh(CP^2, \tau^1)\}$
$\tau: P, product$
1: Inh$(CPS^0, \sigma^1)\ \rightarrow [1]\{Inh(CPSX^2, \sigma^1)\}$
$\sigma: P, product$
2: Inh$(CPS^0, \sigma^1)\ \rightarrow [1]\{Inh(CCRT^2, \sigma^1)\}$
$\sigma: CRT, creator$
3: Inh$(CPSX^0, \tau^1)\ \rightarrow [2]\{Inh(CPS^2, \tau^1)\}$
$\tau: P, product$
4: Inh$(CPSX^0, \sigma^1)\ \rightarrow [3]\emptyset$
$\sigma: P, product$
5: Inh$(\tau^1, P^0)\ \rightarrow [3]\{Inh(\tau^1, product^2)\}$
$\tau: CPS, CP, concreteProduct$
6: Inh$(\sigma^1, CRT^0)\ \rightarrow [1]\{Inh(\sigma^1, creator^2)\}$
$\tau: CPS, CCRT, concreteCreator$
7: Inh$(CP^0, \tau^1)\ \rightarrow [4]\{Inh(concreteProduct^2, \tau^1)\}$
$\tau: P, product$

8: Ins$(\sigma^1, CP^0)\ \rightarrow [4]\{Ins(\sigma^1, concreteProduct^2)\}$
$\sigma: CCRT, concreteCreator$
9: Inh$(CCRT^0, \tau^1)\ \rightarrow [7]\{Inh(concreteCreator^2, \tau^1)\}$
$\tau: CRT, creator$
10: Ins$(CCRT^0, \sigma^1)\ \rightarrow [7]\{Ins(concreteCreator^2, \sigma^1)\}$
$\sigma: CP, concreteProduct$

3.6. Patrón Composite

Forma parte de los patrones estructurales. La intención de este patrón de diseño se enfoca a encapsular los objetos en jerarquías y tratar dicha jerarquía o sus partes como objetos individuales.

3.6.1 Diagrama de clases estático

El diagrama de clases del patrón de diseño *Composite* en su forma estática es el siguiente:

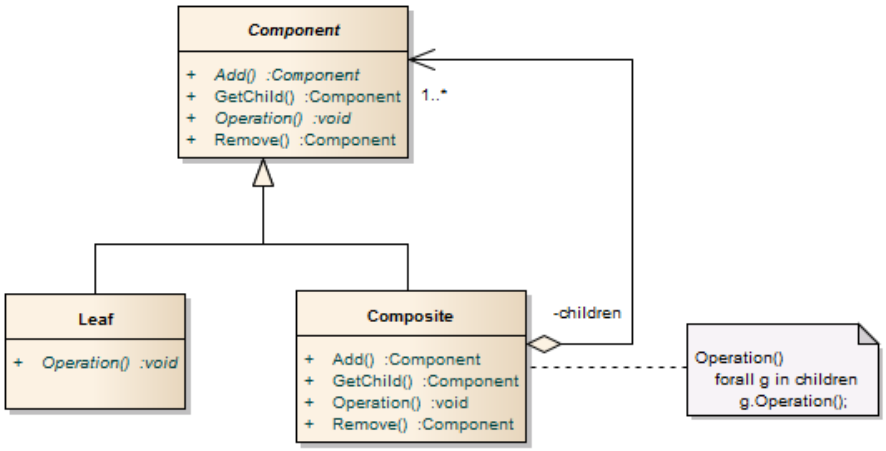

Figura 12. Patrón de diseño Composite en su representación Estática (Pérez, 2014)

3.6.2 Diagrama de clases dinámico

El diagrama de clases del patrón de diseño *Composite* en su forma dinámica, es decir, que existen una o más instancias de la clase *Leaf* es el siguiente:

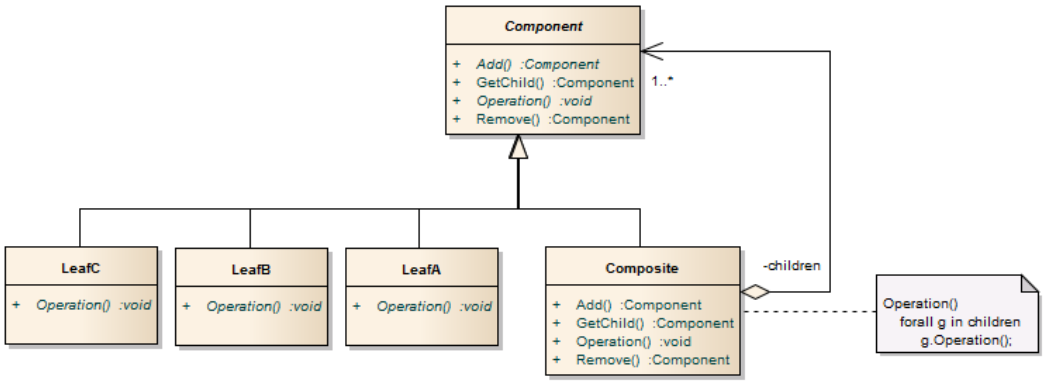

Figura 13. Patrón de diseño Composite en su representación Dinámica (Pérez, 2014)

3.6.3 Roles del patrón Composite

El patrón ***Composite*** define 3 roles diferentes: *Component, Composite y Leaf*. Dentro de esta estructura, podemos encontrar una o más clases que ocupen el rol de *Leaf*.

3.6.4 Definición de la regla gramatical del patrón de diseño Composite estático

CompositePattern = (
{Z, CE, LF, CT},
{component, leaf, composite},
{Inh, Comp},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow <\{CE^2, LF^2, CT^2\}, \{Inh(LF^2, CE^2), Inh(CT^2, CE^2), Comp(CT^2, CE^2)\}>$
1: $CE^0 \rightarrow < \{component^2\}, \emptyset >$
2: $LF^0 \rightarrow < \{leaf^2\}, \emptyset >$
3: $CT^0 \rightarrow < \{composite^2\}, \emptyset >$

R:
0: $Inh(\tau^1, CE^0) \rightarrow [1]\{Inh(\tau^1, component^2)\}$
$\tau: LF, CT, leaf, composite$
1: $Comp(\sigma^1, CE^0) \rightarrow [1]\{Comp(\sigma^1, component^2)\}$
$\sigma: CT, composite$
2: $Inh(LF^0, \tau^1) \rightarrow [2]\{Inh(leaf^2, \tau^1)\}$
$\tau: CE, component$
3: $Inh(CT^0, \sigma^1) \rightarrow [3]\{Inh(composite^2, \sigma^1)\}$
$\sigma: CE, component$
4: $Comp(CT^0, \tau^1) \rightarrow [3]\{Comp(composite^2, \tau^1)\}$
$\tau: CE, component$

3.6.5 Definición de la regla gramatical del patrón de diseño Composite dinámico

CompositePattern = (
{Z, CE, LV, LVX, LF, CT},
{component, leaf, composite},
{Inh, Comp},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow <\{CE^2, LV^2, CT^2\}, \{Inh(LV^2, CE^2), Inh(CT^2, CE^2), Comp(CT^2, CE^2)\}>$
1: $LV^0 \rightarrow <\{LF^2, LVX^2\}, \emptyset >$
2: $LVX^0 \rightarrow <\{LV^2\}>$
3: $LVX^0 \rightarrow < \{\lambda\}, \emptyset >$
4: $CE^0 \rightarrow < \{component^2\}, \emptyset >$
5: $CT^0 \rightarrow < \{composite^2\}, \emptyset >$
6: $LF^0 \rightarrow < \{leaf^2\}, \emptyset >$

R:
0: $Inh(LV^0, \tau^1) \rightarrow [1]\{Inh(LF^2, \tau^1), Inh(LVX^2, \tau^1)\}$
$\tau: CE, component$
1: $Inh(LVX^0, \sigma^1) \rightarrow [2]\{Inh(LV^2, \sigma^1)\}$
$\sigma: CE, component$
2: $Inh(LVX^0, \tau^1) \rightarrow [3]\emptyset$

$\tau: CE, component$

3: Inh$(\sigma^1, CE^0) \rightarrow [4]\{Inh(\sigma^1, component^2)\}$

$\sigma: LV, LVX, LF, CT, leaf, composite$

4: Comp$(\sigma^1, CE^0) \rightarrow [4]\{Comp(\sigma^1, component^2)\}$

$\sigma: CT, composite$

5: Inh$(CT^0, \tau^1) \rightarrow [5]\{Inh(composite^2, \tau^1)\}$

$\tau: CE, component$

6: Comp$(CT^0, \sigma^1) \rightarrow [5]\{Comp(composite^2, \sigma^1)\}$

$\sigma: CE, component$

7: Inh$(LF^0, \tau^1) \rightarrow [6]\{Inh(leaf^2, \tau^1)\}$

$\tau: CE, component$

3.7. Combinación Strategy – Template Method

3.7.1 Roles de la combinación Strategy – Template Method

En esta combinación de patrones participan 5 roles:

Los 3 roles del patrón *Strategy*:

- *Context*
- *Strategy*
- *ConcreteStrategy*

Los 2 roles del patrón *Template Method*:

- *AbstractClass*
- *ConcreteClass*

3.7.2 Diagrama de clases

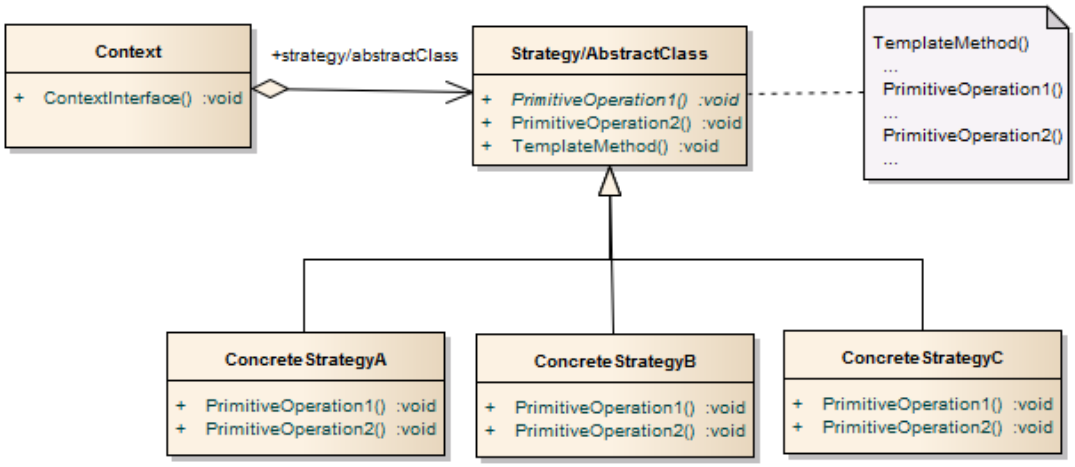

Figura 14. Diagrama de clases de la combinación entre los patrones de diseño Strategy y Template Method (Pérez, 2014)

3.7.3 Definición de la regla gramatical de la combinación Strategy-Template Method

Strategy-TemplateMethod = {
{Z, PSPTM, CX, ST-AC, CSTD-CCD, CST-CC, CSTDX-CCDX},
{context, strategy-abstractClass, concreteStrategy-concreteClass},
{Inh, Comp},
Z,
P,
R
}

P:
0: $Z^0 \rightarrow < \{PSPTM^2\}, \emptyset >$
1: $PSPTM^0 \rightarrow < \{CX^2, ST-AC^2, CSTD-CCD^2\}, \{Inh(CSTD-CCD^2, ST-AC^2), Comp(CX^2, ST-AC^2)\} >$
2: $CSTD\text{-}CCD^0 \rightarrow < \{CST-CC^2, \text{CSTDX} - \text{CCDX}^2\}, \emptyset >$
3: $CSTDX\text{-}CCDX^0 \rightarrow < \{CSTD-CCD^2\}, \emptyset >$
4: $CSTDX\text{-}CCDX^0 \rightarrow < \{\lambda\}, \emptyset >$
5: $CX^0 \rightarrow < \{context^2\}, \emptyset >$

6: $ST\text{-}AC^0 \rightarrow < \{strategy-abstractClass^2 = \{x | \forall x(C(x) \wedge M(x) \wedge Ma(x) \wedge Mt(x))\}\}, \emptyset >$
7: $CST\text{-}CC^0 \rightarrow < \{concreteStrategy-concreteClass^2 = \{x | \forall x(C(x) \wedge Mi(x) \wedge \neg Ma(x)\}\}, \emptyset >$

Dónde:
C(x) = “x es una clase”
Mi(x) = “x tiene métodos implementados”
Ma(x) = “x tiene al menos un método abstracto”
M(x) = “x tiene métodos”
Mt(x) = “x tiene un método TemplateMethod”

R:
0: $\text{Inh}(CSTD-CCD^0, \tau^1) \rightarrow [2]\{Inh(CST-CC^2, \tau^1), Inh(\text{CSTDX} - \text{CCDX}^2, \tau^1)\}$
$\tau^1: ST-AC, strategy-abstractClass$
1: $\text{Inh}(\text{CSTDX} - \text{CCDX}^0, \sigma^1) \rightarrow [3]\{Inh(CSTD-CCD^2, \sigma^1)\}$
$\sigma^1: ST-AC, strategy-abstractClass$
2: $\text{Inh}(\text{CSTDX} - \text{CCDX}^0, \tau^1) \rightarrow [4]\ \emptyset$
$\tau^1: ST-AC, strategy-abstractClass$
3: $\text{Comp}(\text{CX}^0, \sigma^1) \rightarrow [5]\{Comp(context^2, \sigma^1)\}$
$\sigma^1: ST, strategy$
4: $\text{Inh}(\tau^1, S-AC^0) \rightarrow [6]\{Inh(\tau^1, strategy-abstractClass^2)\}$
$\tau^1: CSTD-CCD, concreteStrategy-concreteClass$
5: $\text{Comp}(\sigma^1, S-AC^0) \rightarrow [6]\{Comp(\sigma^1, strategy-abstractClass^2)\}$
$\sigma^1: CX, context$
6: $\text{Inh}(CST-CC^0, \tau^1) \rightarrow [7]\{Inh(concreteStrategy-concreteClass^2, \tau^1)\}$
$\tau^1: ST-AC, strategy-abstractClass$

3.8. Combinación Strategy – Composite

3.8.1 Roles de la combinación Strategy – Composite

En esta combinación de patrones participan 6 roles:

Los 3 roles del patrón *Strategy*:

- *Context*
- *Strategy*
- *ConcreteStrategy*

Los 3 roles del patrón *Composite*:

- *Component*
- *Composite*
- *Leaf*

3.8.2 Diagrama de clases

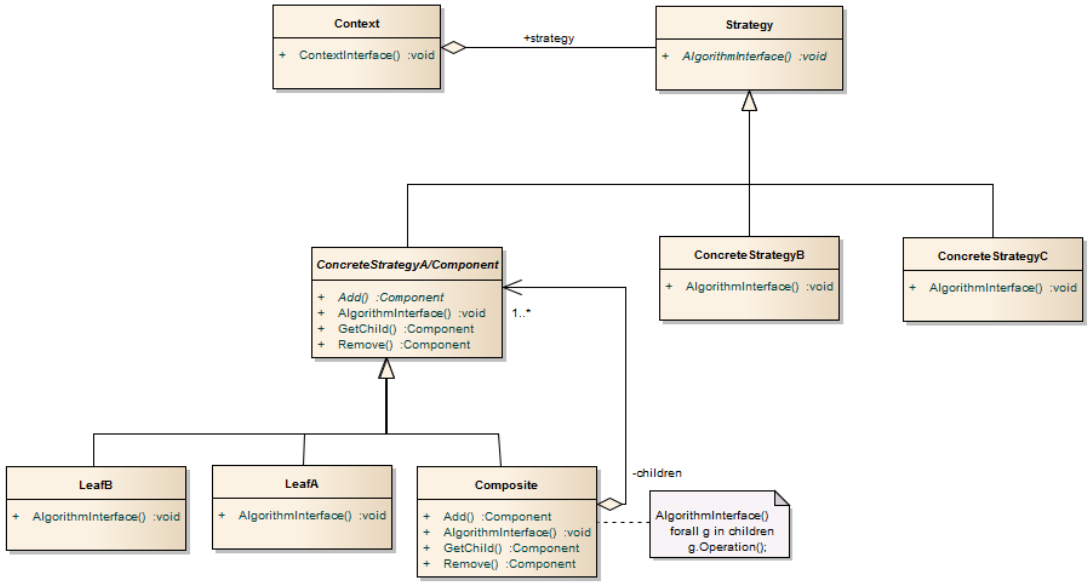

Figura 15. Diagrama de clases de la combinación entre los patrones de diseño Strategy y Composite (Pérez, 2014)

3.8.3 Definición de la regla gramatical de la combinación Strategy-Composite

Strategy-Composite = {
{Z, PS, PC, CX, ST, CSST, CSSTX, CST, CE, LV, LVX, CT, LF},
{context, strategy, concreteStrategy, component, composite, leaf},
{Inh,Comp},
Z,
P,
R
}

P:

0: $Z^0 \rightarrow <\{PS^2, PC^2\}, \{Inh(PC^2, PS^2)\}>$
1: $PS^0 \rightarrow <\{CX^2, ST^2, CSST^2\}, \{Inh(CSST^2, ST^2), Comp(CX^2, ST^2)\}>$
2: $PC^0 \rightarrow <\{CE^2, LV^2, CT^2\}, \{Inh(LV^2, CE^2), Inh(CT^2, CE^2), Comp(CT^2, CE^2)\}>$
3: $CX^0 \rightarrow <\{context^2\}, \emptyset>$
4: $ST^0 \rightarrow <\{strategy^2\}, \emptyset>$
5: $CSST^0 \rightarrow <\{CST^2, CSSTX^2\}, \emptyset>$
6: $CSSTX^0 \rightarrow <\{CSST^2\}, \emptyset>$
7: $CSSTX^0 \rightarrow <\{\lambda\}, \emptyset>$

8: CST^0 → < $\{concreteStrategy^2\}$, Ø >
9: CE^0 → < $\{component^2\}$, Ø >
10: LV^0 → <$\{LF^2, LVX^2\}$, Ø >
11: LVX^0 → <$\{LV^2\}$ >
12: LVX^0 → < $\{\lambda\}$, Ø >
13: CT^0 → < $\{composite^2\}$, Ø >
14: LF^0 → < $\{leaf^2\}$, Ø >

R:

0: Inh(τ^1, PS^0) → [1]$\{Inh(\tau^1, ST^2)\}$
τ: $PC, CE, component$
1: Inh(PC^0, σ^1) → [2]$\{Inh(CE^2, \sigma^1)\}$
σ: $PC, ST, Strategy$
2: $Comp(CX^0, \tau^1)$ → [3]$\{Inh(context^2, \tau^1)\}$
τ: $ST, strategy$
3: Inh(σ^1, ST^0) → [4]$\{Inh(\sigma^1, strategy^2)\}$
σ: $CSST, CST, concreteStrategy, PC, CE, component$
4: $Comp(\tau^1, ST^0)$ → [4]$\{Comp(\tau^1, strategy^2)\}$
τ: $CX, context$
5: Inh$(CSST^0, \sigma^1)$ → [5]$\{Inh(CST^2, \sigma^1), Inh(CSSTX^2, \sigma^1)\}$
σ: $ST, Strategy$
6: Inh$(CSSTX^0, \tau^1)$ → [6]$\{Inh(CSST^2, \tau^1)\}$
τ: $ST, strategy$
7: Inh$(CSSTX^0, \sigma^1)$ → [7]Ø
σ: $ST, Strategy$
8: Inh(CST^0, τ^1) → [8]$\{Inh(concreteStrategy^2, \tau^1)\}$
τ: $ST, strategy$
9: Inh(CE^0, σ^1) → [9]$\{Inh(component^2, \sigma^1)\}$
σ: $PS, ST, Strategy$
10: Inh(σ^1, CE^0) → [9]$\{Inh(\sigma^1, component^2)\}$
σ: $LV, LVX, LF, CT, leaf, composite$
11: Comp(σ^1, CE^0) → [9]$\{Comp(\sigma^1, component^2)\}$
σ: $CT, composite$
12: Inh(LV^0, τ^1) → [10]$\{Inh(LF^2, \tau^1), Inh(LVX^2, \tau^1)\}$
τ: $CE, component$
14: Inh(LVX^0, τ^1) → [11]$\{Inh(LV^2, \tau^1)\}$
τ: $CE, component$
16: Inh(LVX^0, τ^1) → [12]Ø
τ: $ST, strategy$
17: Inh(CT^0, τ^1) → [13]$\{Inh(composite^2, \tau^1)\}$
τ: $CE, component$
18: Comp(CT^0, σ^1) → [13]$\{Comp(composite^2, \sigma^1)\}$
σ: $CE, component$
19: Inh(LF^0, σ^1) → [14]$\{Inh(leaf^2, \sigma^1)\}$
σ: $CE, component$

3.9. Combinación Composite-Template Method

3.9.1 Roles de la combinación Composite – Template Method

En esta combinación de patrones participan 6 roles:

Los 3 roles del patrón *Composite*:

- *Component*
- *Composite*
- *Leaf*

Los 2 roles del patrón *Template Method*:

- *AbstractClass*
- *ConcreteClass*

3.9.2 Diagrama de clases

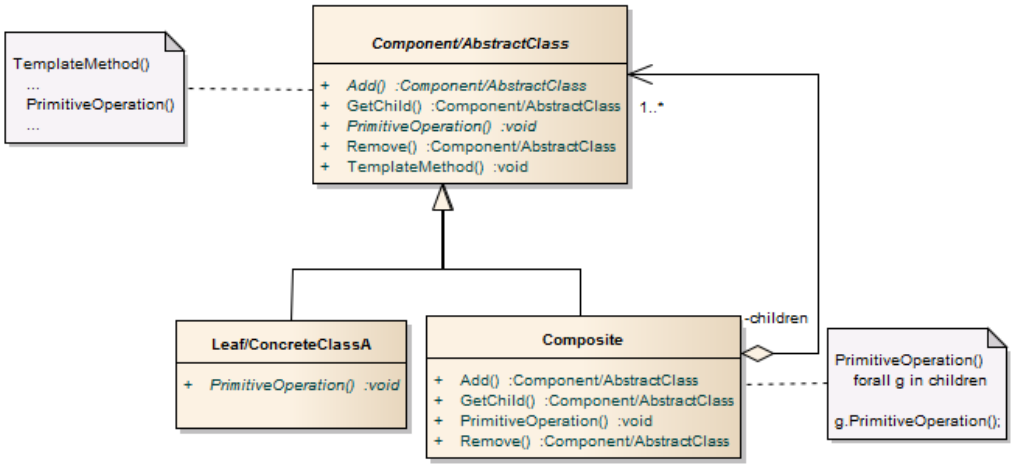

Figura 16. Diagrama de clases de la combinación entre los patrones de diseño Composite y Template Method (Pérez, 2014)

3.9.3 Definición de la regla gramatical de la combinación Composite – Template Method

CompositePattern-TemplateMethod = (
{Z, LFD-CCD, LFDX-CCDX, CE-AC, LF-CC, CT-CC},
{component-abstractClass, leaf-concreteClass, composite-concreteClass},
{Inh, Comp},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow < \{CE - AC^2, LFD - CCD^2, CT - CC^2\}, \{Inh(LFD - CCD^2, CE - AC^2), Inh(CT - CC^2, CE - AC^2), Comp(CT - CC^2, CE - AC^2)\} >$
1: LFD-CCD$^0 \rightarrow < \{LF - CC^2, LFDX - CCDX^2\}, \emptyset >$
2: LFDX-CCDX$^0 \rightarrow < \{CCD^2\}, \emptyset >$
3: LFDX-CCDX$^0 \rightarrow < \{\lambda\}, \emptyset >$
4: CE-AC$^0 \rightarrow < \{component - abstractClass^2 = \{x | \forall x(C(x) \wedge M(x) \wedge Ma(x) \wedge Mt(x))\}\}, \emptyset >$
5: CT-CC$^0 \rightarrow < \{composite - concreteClass^2 = \{x | \forall x(C(x) \wedge Mi(x) \wedge \neg Ma(x))\}\}, \emptyset >$
6: LF-CC$^0 \rightarrow < \{leaf - concreteClass^2 = \{x | \forall x(C(x) \wedge Mi(x) \wedge \neg Ma(x))\}\}, \emptyset >$

Dónde:
C(x) = “x es una clase”
Mi(x) = “x tiene métodos implementados”

Ma(x) = “x tiene al menos un método abstracto”
M(x) = “x tiene métodos”
Mt(x) = “x tiene un método TemplateMethod”

R:
0: Inh$(LFD - CCD^0, \tau^1) \rightarrow [1]\{Inh(LF - CC^2, \tau^1), Inh(LFDX - CCDX^2, \tau^1)\}$
$\tau: CE - AC, component - abstractClass$
1: Inh$(LFDX - CCDX^0, \sigma^1) \rightarrow [2]\{Inh($
$tClass$
2: Inh$(LFDX - CCDX^0, \tau^1) \rightarrow [3]\emptyset$
$\tau: CE - AC, component - abstractClass$
3: Inh$(\sigma^1, CE - AC^0) \rightarrow [4]\{Inh(\sigma^1, component - abstractClass^2)\}$
$\sigma: LFD - CCD, LF - CC, CT - CC, leaf - concreteClass, composite - concreteClass$
4: Comp$(\tau^1, CE - AC^0) \rightarrow [4]\{Comp(\tau^1, component - abstractClass^2)\}$
$\tau: CT - CC, composite - abstractClass$
5: Inh$(CT - CC^0, \sigma^1) \rightarrow [5]\{Inh(composite - concreteClass^2, \sigma^1)\}$
$\sigma: CE - AC, component - abstractClass$
6: Comp$(CT - CC^0, \tau^1) \rightarrow [5]\{Comp(composite - concreteClass^2, \tau^1)\}$
$\tau: CE - AC, component - abstractClass$
7: Inh$(LF - CC^0, \sigma^1) \rightarrow [6]\{Inh(leaf - concreteClass^2, \sigma^1)\}$
$\sigma: CE - AC, component - abstractClass$

3.10. Combinación Strategy-Factory Method

3.10.1 Roles de la combinación Strategy – Factory Method

En esta combinación de patrones participan 7 roles:

Los 3 roles del patrón *Strategy*:

- *Context*
- *Strategy*
- *ConcreteStrategy*

Los 4 roles del patrón *Factory Method:*

- *Creator*
- *ConcreteCreator*
- *Product*
- *ConcreteProduct*

3.10.2 Diagrama de clases

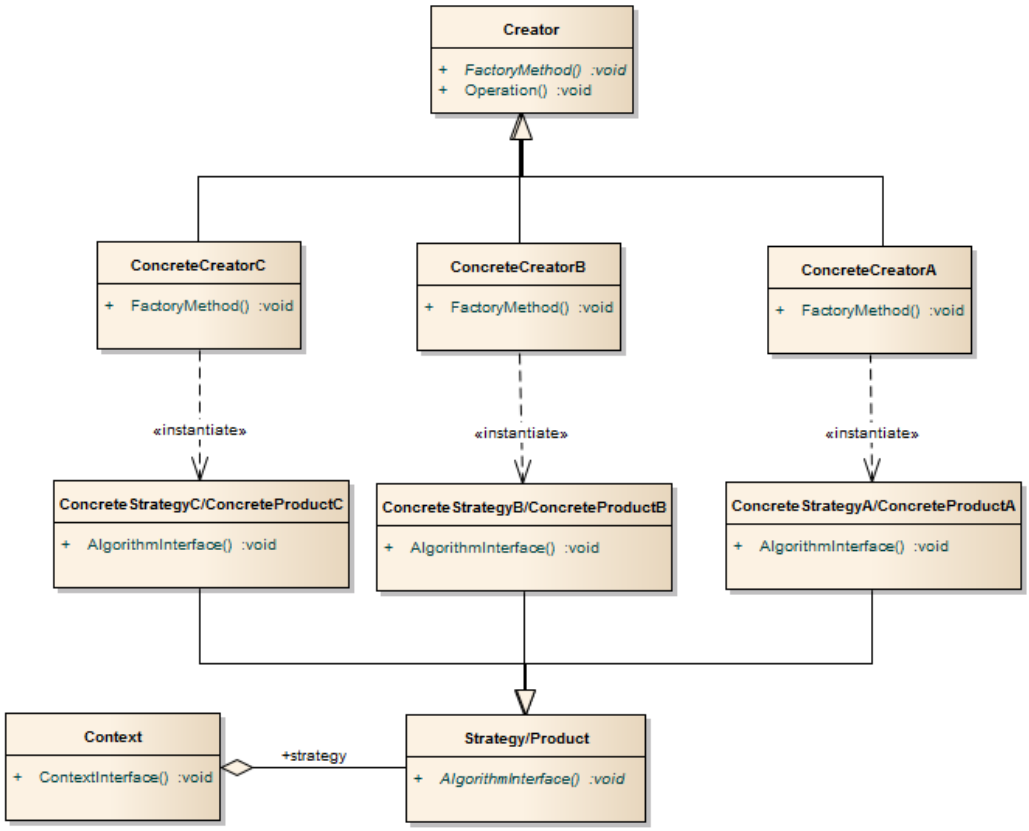

Figura 17. Diagrama de clases de la combinación entre los patrones de diseño Strategy y Factory Method (Pérez, 2014)

3.10.3 Definición de la regla gramatical de la combinación Strategy-Factory Method

Strategy-FactoryMethod = (
{Z, PS, PFM, CPS, CPSX, STP, CRT, CSTCP, CCRT, CX},
{strategy-product, concreteStrategy-concreteProduct, creator, concreteCreator, context},
{Inh, Ins, Comp},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow < \{PS^2, PFM^2\}, \{Comp(PS^2, PFM^2)\} >$
1: $PS^0 \rightarrow < \{CX^2\}, \emptyset >$
2: $PFM^0 \rightarrow < \{STP^2, CRT^2, CPS^2\}, \{Inh(CPS^2, STP^2), Inh(CPS^2, CRT^2)\} >$
3: $CPS^0 \rightarrow < \{CSTCP^2, CPSX^2, CCRT^2\}, \{Ins(CCRT^2, CSTCP^2)\} >$
4: $CPSX^0 \rightarrow < \{CPS^2\}, \emptyset >$
5: $CPSX^0 \rightarrow < \{\lambda\}, \emptyset >$
6: $STP^0 \rightarrow < \{strategy - product^2\}, \emptyset >$
7: $CRT^0 \rightarrow < \{creator^2 = \{x | \forall x(C(x) \wedge M(x) \wedge Ma(x))\}\}, \emptyset >$

8: CSTCP^0 → < $\{concreteStrategy - concreteProduct^2\}, \emptyset$ >
9: CCRT^0 → < $\{concreteCreator^2 = \{x | \forall x(C(x) \wedge Mi(x) \wedge Mf(x) \wedge \neg Ma(x)\}\}, \emptyset$ >
10: CX^0 → < $\{context^2\}, \emptyset$ >

Dónde:

C(x) = “x es una clase”
Mi(x) = “x tiene métodos implementados”
Ma(x) = “x tiene un método abstracto”
M(x) = “x tiene o no método”
Mf(x) = “x tiene un método Factory Method”

R:
0: Comp$(PS^0, \tau^1) \rightarrow [1]\{Comp(CX^2, \tau^1)\}$
$\tau: PFM, STP, strategy - product$
1: $Comp(\sigma^1, ST^0) \rightarrow [2]\{Comp(\sigma^1, STP^2)\}$
$\sigma: CX, context$
2: Inh$(CPS^0, \tau^1) \rightarrow [3]\{Inh(CP^2, \tau^1)\}$
$\tau: STP, strategy - product$
3: Inh$(CPS^0, \sigma^1) \rightarrow [3]\{Inh(CPSX^2, \sigma^1)\}$
$\sigma: STP, strategy - \ product$
4: Inh$(CPS^0, \tau^1) \rightarrow [3]\{Inh(CCRT^2, \tau^1)\}$
$\tau: C, creator$
5: Inh$(CPSX^0, \sigma^1) \rightarrow [4]\{Inh(CPS^2, \sigma^1)\}$
$\sigma: STP, strategy - product$
6: Inh$(CPSX^0, \tau^1) \rightarrow [5]\emptyset$
$\sigma: STP, strategy - product$
7: Inh$(\tau^1, STP^0) \rightarrow [6]\{Inh(\tau^1, strategy - product^2)\}$
$\tau: CPS$, CSTCP, $concreteStrategy - concreteProduct$
8: Inh$(\sigma^1, CRT^0) \rightarrow [7]\{Inh(\sigma^1, creator^2)\}$
$\tau: CPS, CCRT, concreteCreator$
9: Inh$(CSTCP^0, \tau^1) \rightarrow [8]\{Inh(concreteStrategy - concreteProduct^2, \tau^1)\}$
$\tau: STP, strategy - product$
10: Ins$(\sigma^1, CSTCP^0) \rightarrow [8]\{Ins(\sigma^1, concreteStrategy - concreteProduct^2)\}$
$\sigma: CCRT, concreteCreator$
11: Inh$(CCRT^0, \tau^1) \rightarrow [9]\{Inh(concreteCreator^2, \tau^1)\}$
$\tau: CRT, creator$
12: Ins$(CCRT^0, \tau^1) \rightarrow [9]\{Ins(concreteCreator^2, \tau^1)\}$
$\tau: CSTCP, concreteStrategy - concreteProduct$
13: $Comp(CX^0, \sigma^1) \rightarrow [10]\{Comp(context^2, \sigma^1)\}$
$\sigma: PFM, STP, strategy - product$

3.11. Combinación de Template Method – Factory Method

3.11.1 Roles de la combinación Template Method – Factory Method

En esta combinación de patrones participan 6 roles:

Los 2 roles del patrón *Template Method*:

- *AbstractClass*
- *ConcreteClass*

Los 4 roles del patrón *Factory Method*:

- *Creator*
- *ConcreteCreator*
- *Product*
- *ConcreteProduct*

3.11.2 Diagrama de clases

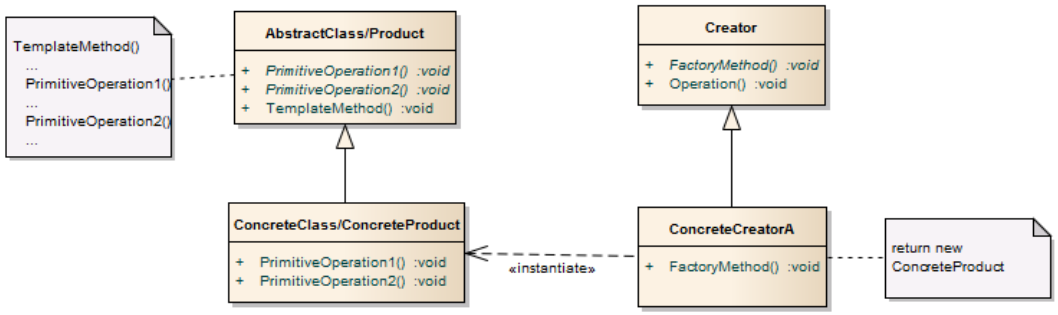

Figura 18. Diagrama de clases de la combinación entre los patrones de diseño Template Method y Factory Method (Pérez, 2014)

3.11.3 Definición de la regla gramatical de la combinación Template method – Factory Method

TemplateMethod-FactoryMethod = (
{Z, PTM-PFM, CPS, CPSX, ACP, CCCP, CRT, CCRT},
{abstractClass-product, concreteClass-concreteProduct,creator,concreteCreator },
{Inh,Ins},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow < \{PTM - PFM^2\}, \emptyset >$
1: $PTM - PFM^0 \rightarrow < \{\mathrm{ACP}^2, \mathrm{CRT}^2, CPS^2\}, \{Inh(CPS^2, ACP^2)\}, \{Inh(CPS^2, CRT^2)\}$ Ø>
2: $\mathrm{CPS}^0 \rightarrow <\{CCCP^2, CPSX^2, CCRT^2\}, \{Ins(CCRT^2, CCCP^2)\} >$

3: CPSX^0 → <$\{CPS^2\}$ >
4: CPSX^0 → < $\{\lambda\}, \emptyset$ >
5: ACP^0 → < $\{abstractClass - product^2 = \{x|\forall x(C(x)\text{^}M(x)\text{^ }Ma(x)\text{^ }Mt(x))\}\}, \emptyset$ >
6: CRT^0 → < $\{creator^2 = \{x|\forall x(C(x)\text{^}M(x)\text{^ }Ma(x))\}\}$, $\emptyset$ >
7: CCCP^0 → < $\{concreteClass - concreteProduct^2 = \{x|\forall x(C(x)\ \text{^}\ Mi(x)\ \text{^}\ \neg Mfa(x)\ \}\}, \emptyset$ >
8: CCRT^0 → < $\{concreteCreator^2 = \{x|\forall x(C(x)\ \text{^}\ Mi(x)\text{^ }Mf(x)\ \text{^}\ \neg Mfa(x)\ \}\}$, $\emptyset$ >

Dónde:
C(x) = “x es una clase”
Mi(x) = “x tiene métodos implementados”
Ma(x) = “x tiene al menos un método abstracto”
M(x) = “x tiene métodos”
Mt(x) = “x tiene un método TemplateMethod”
Mf(x) = “x tiene método Factory Method”

R:

0: Inh(CPS^0, τ^1) → [2]$\{Inh(CCCP^2, \tau^1)\}$
τ: $ACP, abstractClass - product$
1: Inh(CPS^0, σ^1) → [2]$\{Inh(CPSX^2, \sigma^1)\}$
σ: $ACP, abstractClass - product$
2: Inh(CPS^0, σ^1) → [2]$\{Inh(CCRT^2, \sigma^1)\}$
σ: $C, creator$
3: Inh$(CPSX^0, \tau^1)$ → [3]$\{Inh(CPS^2, \tau^1)\}$
τ: $ACP, abstractClass - product$
4: Inh$(CPSX^0, \sigma^1)$ → [4]$\emptyset$
σ: $ACP, abstractClass - product$
5: Inh(τ^1, ACP^0) → [5]$\{Inh(\tau^1, product^2)\}$
τ: $CPS, CP, concreteProduct$
6: Inh(σ^1, CRT^0) → [6]$\{Inh(\sigma^1, creator^2)\}$
τ: $CPS, CCRT, concreteCreator$
7: Inh$(CCCP^0, \tau^1)$ → [7]$\{Inh(concreteProduct^2, \tau^1)\}$
τ: $ACP, abstractClass - product$
8: Ins$(\sigma^1, CCCP^0)$ → [7]$\{Ins(\sigma^1, concreteProduct^2)\}$
σ: $CCRT, concreteCreator$
9: Inh$(CCRT^0, \tau^1)$ → [8]$\{Inh(concreteCreator^2, \tau^1)\}$
τ: $CRT, creator$
10: Ins$(CCRT^0, \sigma^1)$ → [8]$\{Ins(concreteCreator^2, \sigma^1)\}$
σ: $CCCP, concreteClass - concreteProduct$

3.12. Combinación Composite – Factory Method

3.12.1 Roles de la combinación Composite – Factory Method

En esta combinación de patrones participan 6 roles:

Los 3 roles del patrón *Composite*:

- *Component*
- *Composite*
- *Leaf*

Los 4 roles del patrón *Factory Method*:

- *Creator*
- *ConcreteCreator*
- *Product*
- *ConcreteProduct*

3.12.2 Diagrama de clases

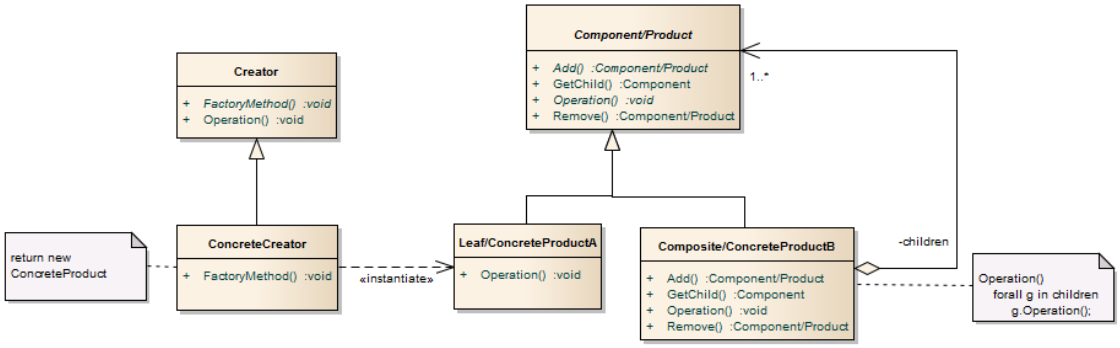

Figura 19. Diagrama de clases de la combinación entre los patrones de diseño Composite y Factory Method (Pérez, 2014)

3.12.3 Definición de la regla gramatical de la combinación Composite-Factory Method

Composite-FactoryMethod = (
{Z, CPLS, CPLSX, CPCS, CPCSX, P, CRTP, CPLF, CCRTP, CRTCT, CPCT, CCRTCT},
{product, creator-product, concreteProduct-Leaf, concreteCreator-product,
creatorComposite , concreteProduct-Composite, concreteCreatorComposte},
{Inh, Ins},
Z,
P,
R
)

P:
0: $Z^0 \rightarrow$
$<\{P^2, \text{CRTP}^2, CPLS^2, CRTCT^2, CPCS^2\}, \{Inh(CPLS^2, P^2), Inh(CPLS^2, CRTP^2), Inh(CPCS^2, P^2), Inh(CPCS^2, CRTCT^2)\}>$
1: $\text{CPLS}^0 \rightarrow <\{CPLF^2, CPLSX^2, \text{CCRTP}^2\}, \{Ins(\text{CCRTP}^2, CPLF^2)\}>$
2: $\text{CPLSX}^0 \rightarrow <\{CPLS^2\}>$
3: $\text{CPLSX}^0 \rightarrow <\{\lambda\}, \emptyset>$
4: $\text{CPCS}^0 \rightarrow <\{CPCT^2, CPCSX^2, CCRTCT^2\}, \{Ins(CCRTCT^2, CPCT^2)\}>$

5: CPCSX$^0 \rightarrow <\{CPCS^2\}>$
6: CPCSX$^0 \rightarrow <\{\lambda\}, \emptyset>$
7: P$^0 \rightarrow <\{product^2\},\ \emptyset>$
8: CRTP$^0 \rightarrow <\{creator - product^2 = \{x|\forall x(C(x)\text{^}M(x)\text{^}\ Mfa(x))\}\},\ \emptyset>$
9: CPLF$^0 \rightarrow <\{concreteProduct - leaf^2\}, \emptyset>$
10: CCRTP$^0 \rightarrow <\{concreteCreator - product^2 = \{x|\forall x(C(x)\ \text{^}\ Mf(x)\ Mi(x)\ \text{^}\ \neg Ma(x)\ \}\}, \emptyset>$
11: CRTCT$^0 \rightarrow <\{creator - composite^2 = \{x|\forall x(C(x)\text{^}M(x)\text{^}\ Mfa(x))\}\},\ \emptyset>$
12: CPCT$^0 \rightarrow <\{concreteProduct - composite^2\}, \emptyset>$
13: CCRTCT$^0 \rightarrow <\{concreteCreator - composite^2 = \{x|\forall x(C(x)\ \text{^}\ Mf(x)\ Mi(x)\ \text{^}\ \neg Ma(x)\ \}\},\ \emptyset>$

Dónde:

C(x) = “x es una clase”
Mi(x) = “x tiene métodos implementados”
Ma(x) = “x tiene al menos un método abstracto”
Mf(x) = “x tiene un método Factory Method”
M(x) = “x tiene o no método”

R:

0: Inh$(CPLS^0, \tau^1) \rightarrow [1]\{Inh(CPLF^2, \tau^1)\}$
$\tau: P, product$
1: Inh$(CPLS^0, \sigma^1) \rightarrow [1]\{Inh(CPLSX^2, \sigma^1)\}$
$\sigma: P, product$
2: Inh$(CPLS^0, \tau^1) \rightarrow [1]\{Inh(CCRTP^2, \tau^1)\}$
$\tau: CRTP, creator - product$
3: Inh$(CPLSX^0, \sigma^1) \rightarrow [2]\{Inh(CPLS^2, \sigma^1)\}$
$\sigma: P, product$
4: Inh$(CPLSX^0, \tau^1) \rightarrow [3]\emptyset$
$\tau: P, product$
5: Inh$(CPCS^0, \sigma^1) \rightarrow [4]\{Inh(CPC^2, \sigma^1)\}$
$\sigma: P, product$
6: Inh$(CPCS^0, \tau^1) \rightarrow [4]\{Inh(CPCSX^2, \tau^1)\}$
$\tau: P, product$
7: Inh$(CPCS^0, \sigma^1) \rightarrow [4]\{Inh(CCRTCT^2, \sigma^1)\}$
$\sigma: CRTCT, ceator - composite$
8: Inh$(CPCSX^0, \tau^1) \rightarrow [5]\{Inh(CPCS^2, \tau^1), Inh(CPCS^2, \tau^1)\}$
$\tau: P, product$
9: Inh$(CPCSX^0, \sigma^1) \rightarrow [6]\emptyset$
$\sigma: P, product$
10: Inh$(\tau^1, P^0) \rightarrow [7]\{Inh(\tau^1, product^2)\}$
$\tau: CPLS, CPLF, CPLSX, CPCS, CCRTP, CCPCSX, CPCT, CCRTCT, concreteProduct - Leaf$
11: Inh$(\sigma^1, CRCT^0) \rightarrow [8]\{Inh(\sigma^1, creator - product^2)\}$
$\tau: CPLS, CCRTP, concreteCreator - Product$
12: Inh$(CPLF^0, \tau^1) \rightarrow [9]\{Inh(concreteProduct - leaf^2, \tau^1)\}$
$\tau: P, product$
13: Ins$(\sigma^1, CPLF^0) \rightarrow [9]\{Ins(\sigma^1, concreteProductLeaf^2)\}$
$\sigma: CCRTP, concreteCreator - product$
14: Inh$(CCRTP^0, \tau^1) \rightarrow [10]\{Inh(concreteCreator - product^2, \tau^1)\}$
$\tau: CRTP, creator - product$
15: Ins$(CCRTP^0, \tau^1) \rightarrow [10]\{Ins(concreteCreator - product^2, \tau^1)\}$
$\tau: CPLF, concreteProduct - leaf$
16: Inh$(\sigma^1, CRTCT^0) \rightarrow [11]\{Inh(\sigma^1, creator - Composite^2)\}$
$\sigma: CPCS, CCRTCT, concreteCreator - composite$
17: Inh$(CPCT^0, \sigma^1) \rightarrow [12]\{Inh(concreteProduct - Composite^2, \sigma^1)\}$
$\sigma: P, product$
18: Ins$(\sigma^1, CPCT^0) \rightarrow [12]\{Ins(\sigma^1, concreteProduct - Composite^2)\}$

σ: $CCRTCT, concreteCreator - composite$

19: Inh($CCRTCT^0, \sigma^1$) → [12]{$Inh(concreteCreator - Composite^2, \sigma^1)$}

σ: $CRTCT, creator - Composite$

20: Ins($CCRTCT^0, \sigma^1$) → [13]{$Ins(concreteCreator - Composite^2, \sigma^1)$}

σ: $CPCT, concreteProduct - Composite$

3.13. Algoritmo aplicado

El algoritmo usado para verificar si un patrón de diseño cumple con las reglas previamente establecidas es tomado del trabajo (Gonzáles, 2010), el cual está basado en un algoritmo LL(1) que fue alterado para recibir un conjunto de s-ítems con sus respectivos r-ítems. Se debe aplicar un ordenamiento de los s-ítems de la cadena de entrada para poder aplicar el algoritmo modificado.

3.13.1 Elementos del algoritmo

s-ítems: Representan a los símbolos de entrada o los roles que participan en cada patrón de diseño. Cuando se combinan patrones de diseño también se combinan sus roles.

Además de los roles que se definen en la gramática, se agregan dos roles necesarios para el algoritmo:

- **$:** representa a un símbolo especial del algoritmo y es usado para verificar cuando una serie de elementos ha iniciado o terminado.

- **VOID:** representa a los símbolos vacíos que no contienen a otros elementos, pero que continúan siendo símbolos.

r-ítems: Representan las relaciones entre las clases de los s-ítem. Los tipos de relaciones son : Herencia(Inh), Instanciación (Ins), Agregación (Agg), Composición(Comp) y vacía (VOID).

Tabla de análisis: Contiene las formas en la que los símbolos no-terminales pueden sustituirse por otros símbolos terminales u otros símbolos no terminales, todo esto dependiendo de las entradas que se tengan en la pilas de entrada. La estructura de la tabla está definida de la siguiente manera: los renglones son representados por un símbolo no terminal, mientras que las columnas son representadas por los roles contenidos por los s-items de entrada. Existe una tabla de análisis para cada patrón y sus combinaciones.

Pila: La pila contiene a los símbolos que se sustituyen y usan. Estos pueden ser símbolos terminales, no terminales o el símbolo especial **$.** Cuando el algoritmo llega a este símbolo especial, indica que todos los demás símbolos han sido consumidos y que la pila se encuentra vacía. El orden en que los símbolos son introducidos o remplazados en la pila es dado por **la tabla de análisis**.

Top: El top es un símbolo que no se consume ni se sustituye, sólo se dedica a indicar el símbolo superior de la pila de símbolos. Esto es el símbolo actual que se esté manejando.

Top es necesario dentro de las condiciones de control para saber qué símbolo se encuentra en la cima de la pila y saber cuál es el siguiente paso dentro del proceso.

Apuntador de entrada: Este es otro símbolo cuyo propósito es indicar el elemento que se encuentra en la cima de la pila de los **s-items** de entrada. Como sucede con **Top**, también se necesita de este apuntador para llevar el control de los símbolos. En la pila de los **s-items** no se derivan los símbolos, sólo se consumen.

a: Símbolo de control que permite saber qué símbolo se encuentra en la cima de la pila de entrada de los **s-items** para hacer comparaciones e indicar qué símbolos se necesita extraer de la tabla de símbolos. *Ejemplo, Extraer producción **[Top, a],** haría que se buscara en la tabla el símbolo apuntado por **Top**, y el **s-item** apuntado por a.*

K: Este es un listado de las relaciones y símbolos que el algoritmo arroja después de haber sustituido los símbolos de entrada. Si al final del algoritmo, se verifica que las relaciones y elementos que pertenecen a esta lista son iguales a los símbolos de entrada (**r-items**), entonces el patrón se considera como verificado.

Una vez que se han definido todos los elementos necesarios para el algoritmo, se procede a realizar el algoritmo siguiente:

*Repetir hasta que **Top** == **$**{*
*   **Top** = Símbolo en la cima de la pila;*
*   **a** = símbolo apuntado por **Apuntador de Entrada**;*
*   Si **Top** == Símbolo terminal o **$** entonces {*
*   Si **Top** == a entonces {*
*   Extraer el elemento superior de la **pila**; //pila.pop(Top)*
*   Extraer el elemento superior de los **s-items**;*
*   Apuntar **Apuntador de Entrada** al nuevo **s-item** en la cima de los s-items;*
*   } Si no {*
*   Sucedió un error porque las pilas no coinciden;*
*   }//fin Si*
*   } Si no y **Top** == símbolo no terminal entonces {*
*   Si existe la producción [Top, a] entonces {*
*   Remover a **Top** de la Pila;*
*   Introducir los símbolos de la producción [**Top**, **a**] en la Pila, en orden inverso;*
*   Si la producción contiene otras relaciones entonces {*
*   Por cada relación encontrada {*
*   Se hace el intercambio de símbolos no terminales por símbolos terminales;*
*   Si se encuentra una relación con dos símbolos terminales {*
*   Marcar la relación como terminada;*

*Si existe una relación igual en la pila de los **r-items** entonces {*
*Remover relacion marcada de los **r-items**;*
} Si no existe entonces {
Sucedió un error porque la relación marcada no es un elemento de entrada;
}//fin Si
}//fin Si
}//fin Por cada
}//fin Si
} Si no existe {
Sucedió un error porque la producción no existe.
}//fin Si
}//fin Si
}//fin Si
*Si **Top** == **$** y la pila de **r-items** está vacía entonces {*
El patrón es válido;
} Si no entonces {
Sucedió un error porque las pilas de entrada no están vacías;
} //fin Si

CAPÍTULO 4: DISEÑO DEL INTÉRPRETE

4.1 Justificación

Las reglas definidas en los trabajos anteriores se tradujeron y transportaron en archivos XML para que la herramienta SiVerPat pueda reconocerlas y usarlas en su algoritmo LL(1) (Salazar, 2013).

Convertir estas reglas manualmente resulta arduo y difícil puesto que se producen archivos de muchas líneas XML por cada patrón de diseño. Considerando que se trata de archivos XML, estos tiene etiquetas de apertura y cierre por lo cual es propenso al error humano y que el archivo no sea aceptado por SiVerPat.

El intérprete se creó para que el usuario cargue las reglas del patrón de diseño o su combinación en un archivo de texto tipo TXT y sea traducido automáticamente a un archivo XML que pueda ser almacenado.

El intérprete indica si la regla tiene errores gramaticales basadas en una gramática desarrollada para generar estos archivos XML de forma correcta.

4.2 Diagrama de casos de uso

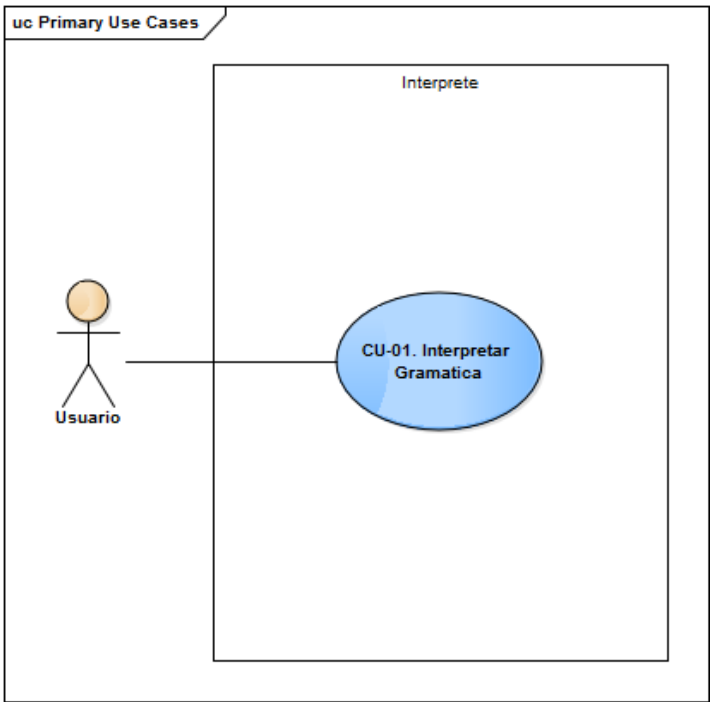

Figura 20. Diagrama de casos de uso

4.4 Diseño

4.4.1 Diagrama de despliegue

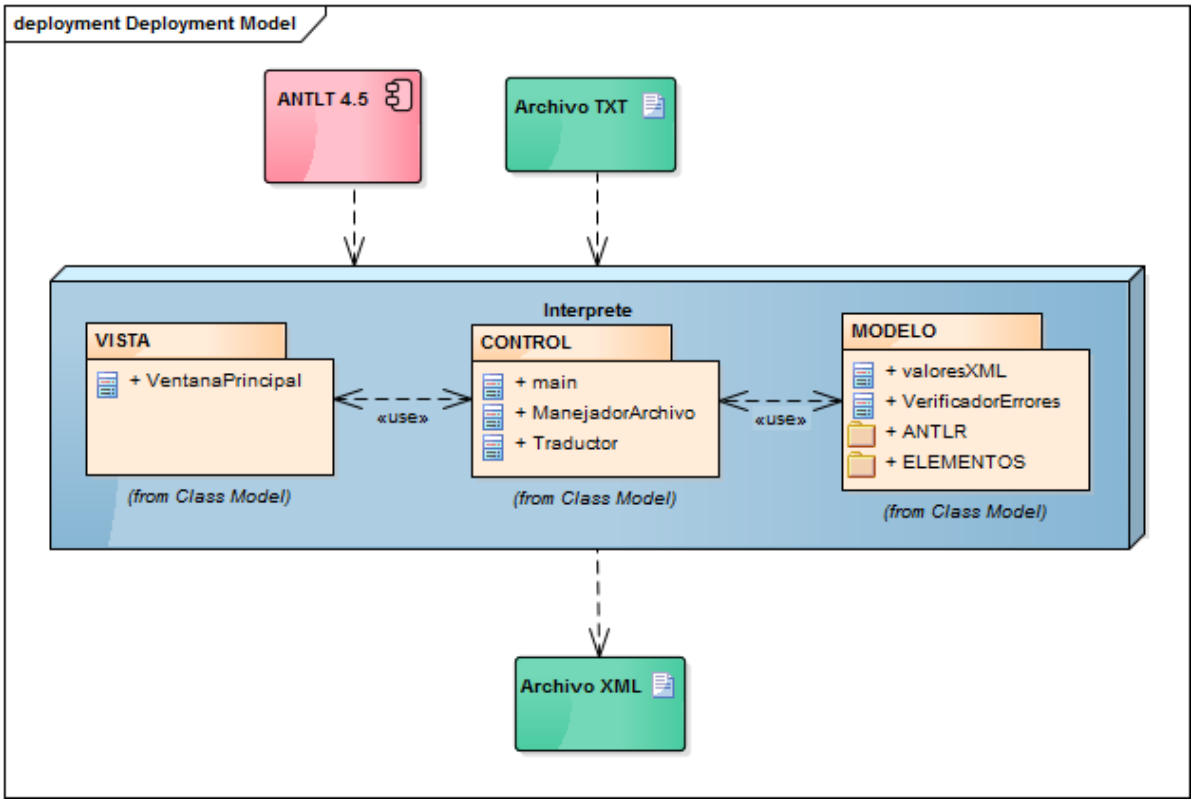

Figura 21. Diagrama de despliegue

4.4.2 Diagrama de paquetes

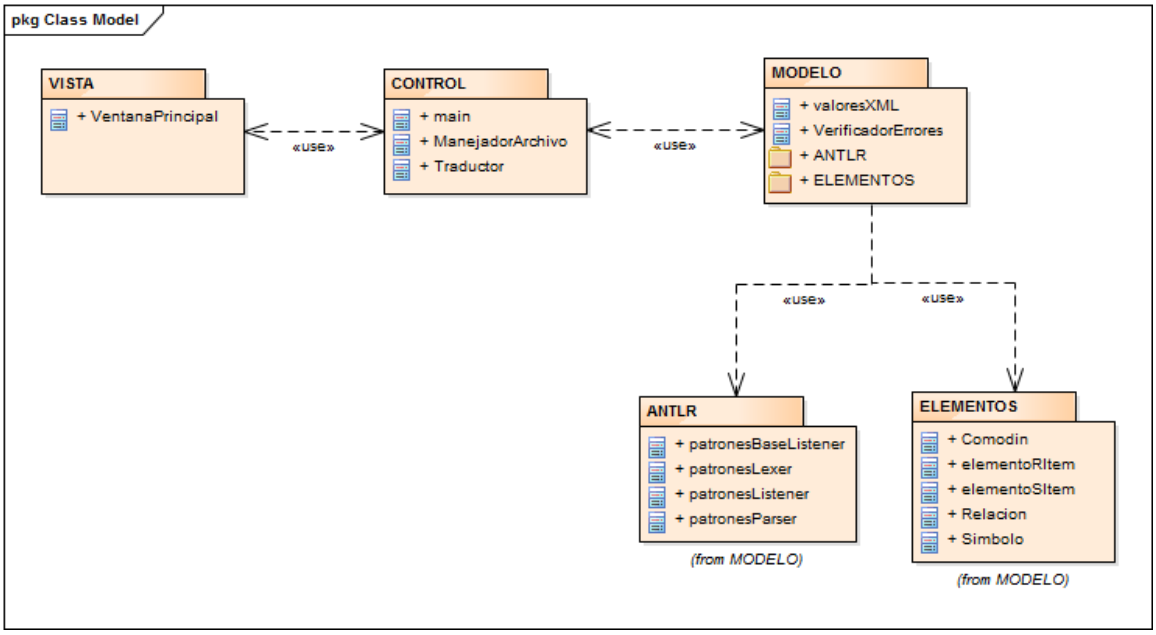

Figura 22. Diagrama de paquetes

4.4.3 Descripción de los Paquetes

Nombre del paquete	Descripción
VISTA	Contiene la interfaz del usuario que consiste en una clase, con una plantilla ligada.
CONTROL	Es el mediador entra la Vista y el Modelo, recibe las acciones del usuario y las dirige hacia las partes del modelo que la necesite.
MODELO	Contiene la lógica del intérprete, desde los elementos que son obtenidos del archivo TXT, la detección de errores y el proceso de traducción.
ANTLR	Contiene las clases generadas por la librería ANTLR 4.5 con la gramática patrones g4, que son usadas para obtener los valores del archivo TXT.
ELEMENTOS	Contiene las clases de los elementos que se obtienen de archivo TXT.

Tabla 2. Descripción de Paquetes

4.4.4 Paquete Vista

Este paquete incluye la clase que muestra la interfaz del usuario a través de la clase VentanaPrincipal.java, que tiene vinculada un archivo *.form* Este es un archivo con formato XML, con extensión *form* que es usado por la herramienta *Intellij IDEA* que almacena el contenido de la interfaz y su posición.

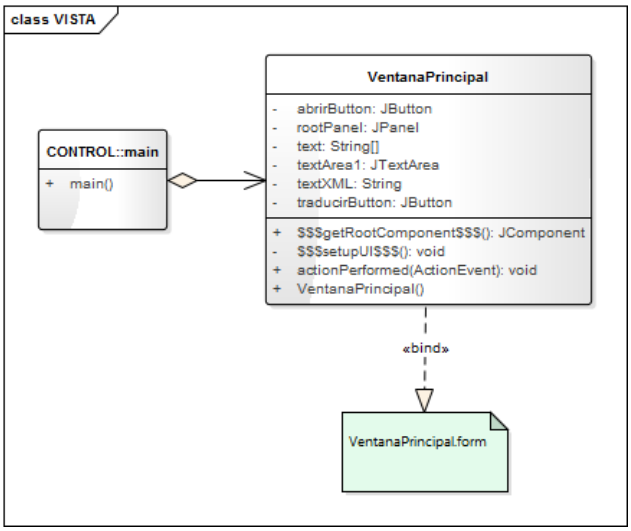

Figura 23. Paquete VISTA

4.4.5 Paquete CONTROL

El paquete CONTROL es el intermediario entre el MODELO y la VISTA, la clase manejadorArchivo es capaz de cargar un archivo y devolver tanto su contenidó como sus características. La clase Traductor recibe el contenido del archivo y usando el paquete MODELO traduce el archivo al formato XML, el cual devuelve a la VentanaPrincipal para que el usuario seleccione el lugar para almacenarlo usando nuevamente la clase manejadorArchivo.

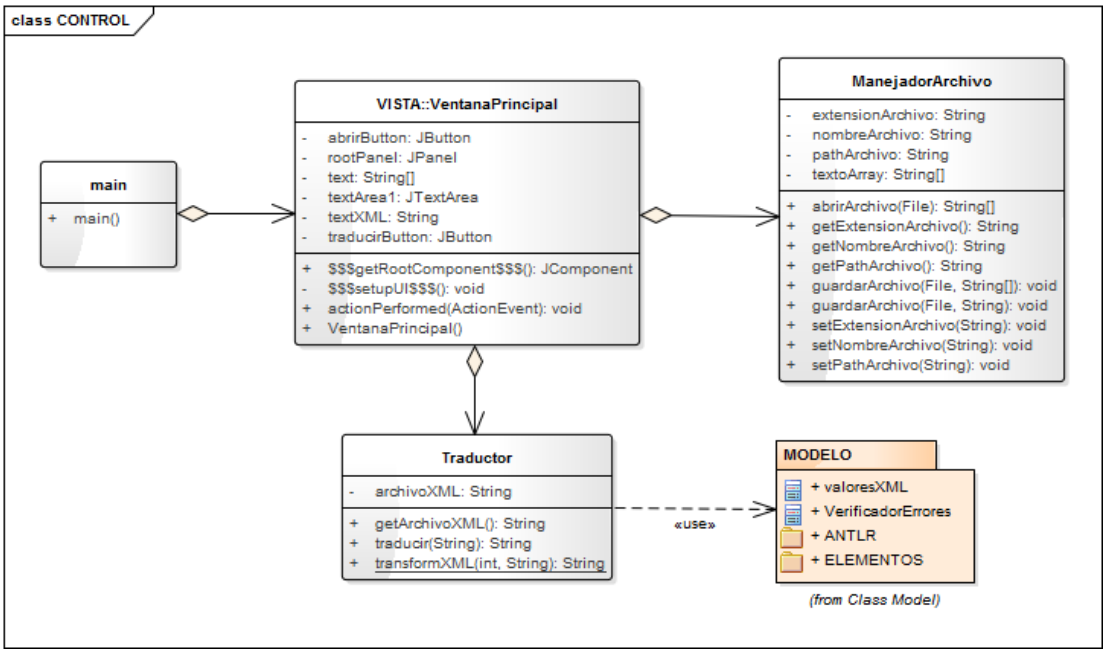

Figura 24. Paquete CONTROL

4.4.6 Paquete MODELO

Este paquete contiene la lógica de la herramienta, la clase verificador de errores notificara si hay algún error con la gramática, mientras que la clase valoresXML convierte el contenido del archivo TXT en XML RAW, primero almacenando los valores en diferentes variables y luego usando esas variables para construir el XML. Al igual que en los paquetes anteriores, se describirán los atributos importantes y se omitirán también los métodos GET y SET de éstos. Además implementa los métodos de la clase patronesBaseListener, que son de la forma enterNode o exitNode, e indican que cuando se entre o salga de algún nodo en la gramática se realizará una acción correspondiente.

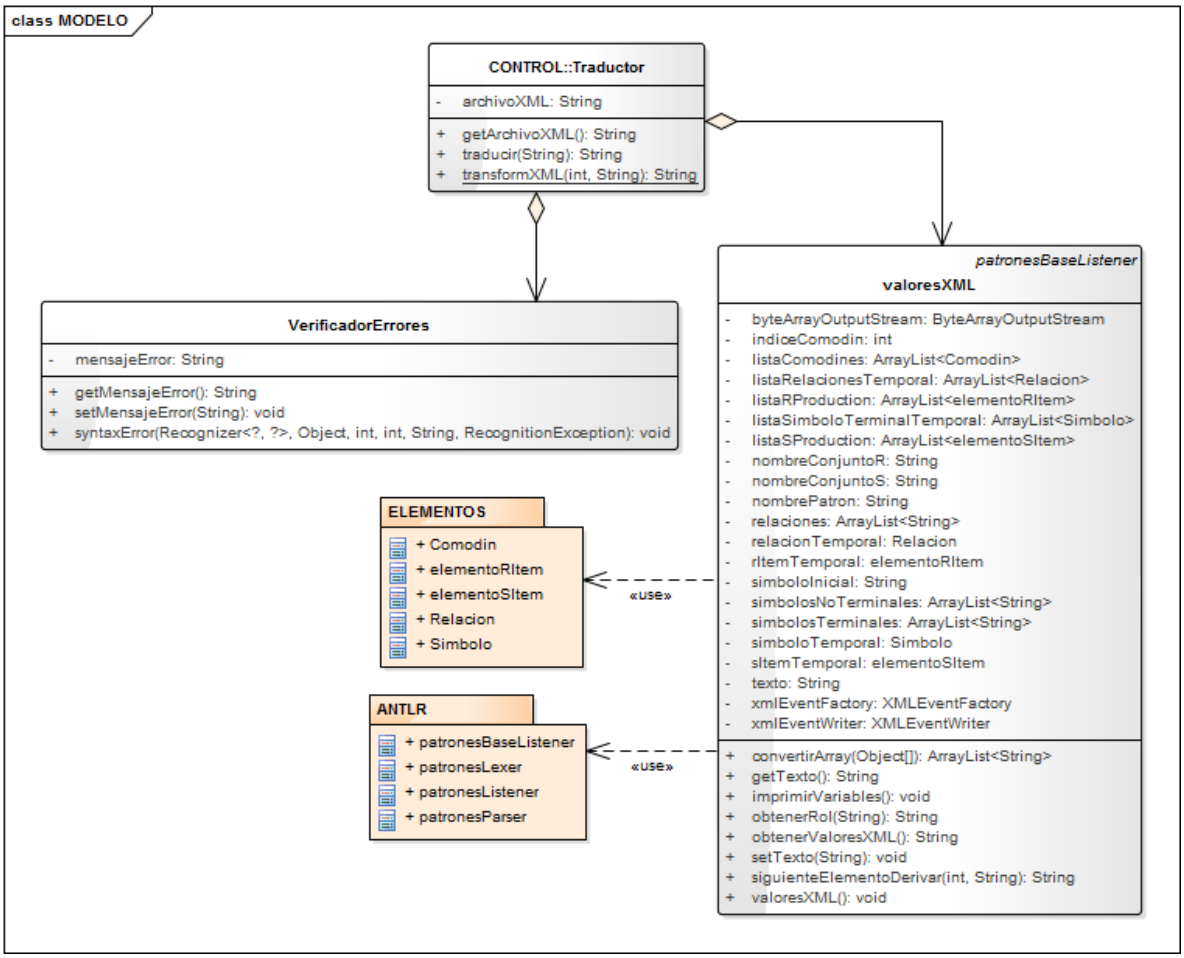

Figura 25. Paquete MODELO

4.4.6.1 Paquete Elementos

Este paquete contiene los elementos del que está compuesta la gramática así como la ubicación donde se van a almacenar sus componentes de forma individual.

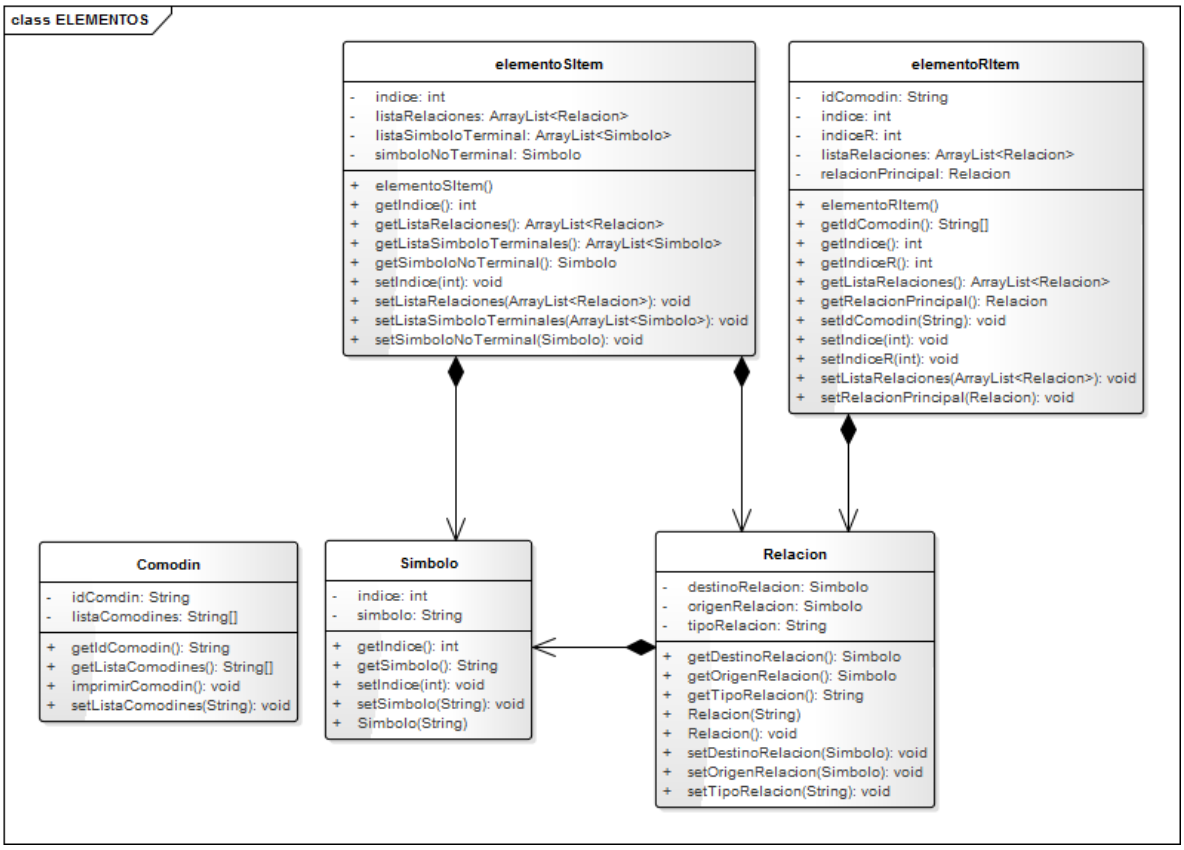

Figura 26. Paquete ELEMENTOS

4.4.6.2 Paquete ANTLR

Este paquete contiene la gramática que es usada por el intérprete para verifcar la redacción correcta del archivo TXT , llamado **patrones.g4**. Con esta gramática la herramienta ANTLR genera el siguiente conjunto de clases y archivos que permiten verificar la construcción gramatical de este archivo.

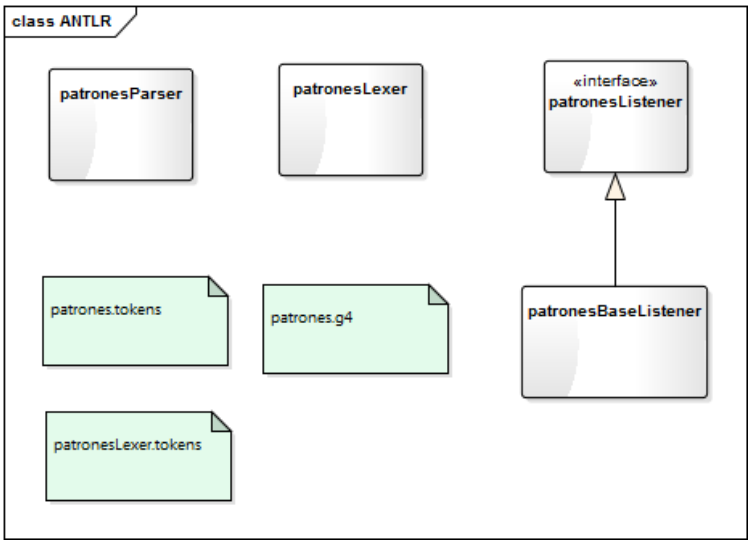

Figura 27. Paquete ANTLR

CAPÍTULO 5: DESARROLLO DEL INTÉRPRETE

5.1 Desarrollo

En esta sección se explica el proceso de desarrollo del intérprete, mostrando la gramática usada para definir la estructura que deberá tener el archivo de texto, en el formato definido para las gramáticas utilizado para la herramienta a lo largo de toda la investigación en sus diferentes etapas, los elementos a modificar o eliminar de esta gramática relacional para que pueda ser reconocido de forma correcta por el intérprete, un ejemplo del patrón de diseño Strategy modificado que es aceptado por la gramática. AL final el proceso seguido por el intérprete para convertir el archivo TXT con las reglas formales en un archivo XML aceptado por la herramienta SiVerPat.

5.1.1 Gramática

La gramática usada por el intérprete es la siguiente:

Inicio → datos sProduccion rProduccion

datos→ nombrePatron '="(' noTerminales ',' terminales',' relaciones ',' inicial',' conjuntoSitems ',' conjuntoRitems')'

nombrePatron→ TEXTO
noTerminales→ '{' TEXTO (',' TEXTO)* '}'
terminales→ '{' TEXTO (',' TEXTO)* '}'
relaciones→ '{' RELACION (',' RELACION)* '}'
inicial→ TEXTO

conjuntoSitems→ TEXTO
conjuntoRitems→ TEXTO

sProduccion→ conjuntoSitems ':' sRegla+
sRegla→ indiceSregla':' simboloNOterminal '-"<'({'simboloTerminal(','simboloTerminal)*'}'|'{'VACIO'}') (',' relacionTerminal '>'|'>')
indiceSregla → NUMERO
simboloNOterminal → VARIABLE

simboloTerminal→ VARIABLE
relacionTerminal→ '{' relacionSola (','relacionSola)*'}'| VACIO

rProduccion→ conjuntoRitems ':' pRegla+
pRegla→ indicepregla':' relacionPrincipal '-' ['indiceR']' relacionesR comodin
indicePregla → NUMERO
indiceR → NUMERO
pSimboloTerminal → TEXTO
pSimboloNoTerminal → TEXTO
relacionPrincipal → RELACION '('VARIABLE(','VARIABLE)* ')'
relacionesR → ('{' relacionsola (','relacionSola)*'}'|VACIO)
relacionsola→ RELACION '('VARIABLE(','VARIABLE)* ')'
comodin→ 'wildcard:' TEXTO(',' TEXTO)*

RELACION→ 'Inh' | 'Comp' | 'Agg' | 'Ins'
VACIO→ 'void'
TEXTO→ [a-z | A-Z]+

NUMERO→ [0-9]+;
VARIABLE→ [a-z|A-Z]+ [0-2]

5.1.1.1 Descripción de las partes de la gramática

Regla	Descripción
Inicio	Regla inicial de la gramática
datos	Obtiene los datos del patrón de diseño
nombrePatron	Indica el nombre del patrón de diseño
noTerminales	Lista de símbolos no terminales
terminales	Lista de símbolos terminales
relaciones	Lista de los tipos de relaciones que se usa el patrón de diseño.
inicial	Indica el símbolo inicial de las producciones S
conjuntoSItems	Indica el símbolo que marca el inicio de las producciones S
conjuntoRItems	Indica el símbolo que marca el inicio de las producciones R
sProduccion	Conjunto de pasos de producción de S- Items
sRegla	Estructura la regla de un S-item de forma individual
indiceSregla	Índice de la regla del S-Item
simboloNOterminal	Símbolo no terminal de la regla del S-Item
simboloTerminal	Símbolo Terminal de la regla del S-item
relacionTerminal	Relación Terminal de la regla del S-item
relacionSola	Estructura de una relación
rProduccion	Conjunto de pasos de la producción de R-Items
pRegla	Estructura de la regla de R-Item
indicePregla	Índice de la Regla R ítem
indiceR	Indice de la relación derivada
pSimboloTerminal	Símbolo terminal de la regla P
pSimboloNoTerminal	Símbolo No terminal de la regla P
relacionPrincipal	Relación principal de la regla R
relacionesR	Conjunto de relaciones en las que se puede derivar la Relacion principal
comodin	Lista de los comodines que puede tomar la regla
RELACION	Tipo de relación, puede ser Inh, Comp, Agg o Ins
VACIO	Indica vacío, se representa con 'void'
TEXTO	Texto que puede incluir mayúsculas y minúsculas
NUMERO	Numero de cualquier tamaño que va desde 0 a 9
VARIABLE	Variable que comienza por una parte Textual y termina con un índice, que es 0,1 o 2

Tabla 3. Descripción de la gramática

El archivo TXT que es cargado contiene las reglas del patrón de diseño a convertir en formato XML para la herramienta SiVerPat.

5.1.1.2 Formato en el archivo TXT

Durante la creación del archivo TXT, fue necesario modificar algunos elementos de la gramática relacional, debido a que contienen elementos que no son reconocidos por el intérprete.

Carácter no reconocido	Remplazar por	Nota
⟦		Eliminar
⟧		Eliminar
→	-	
Ø, ∅	void	
λ	void	
σ	wildcard	
τ	wildcard	
^		Eliminar

Tabla 4. Tabla que muestra los caracteres a remplazar

El nombre del patrón fue cambiado para que solo incluya el nombre de este, por lo tanto "StrategyPattern" se cambió por "Strategy".

Adicionalmente, los patrones Factory Method y Template Method especifican un comportamiento que SiVerPat no es capaz de representar por lo tanto se deben eliminar las pares indicadas, por ejemplo.

TamplateMethodPattern = (
{Z, AC, CC},
{abstractClass, concreteClass},
{Inh},
Z,
P,
R
)

P:

0: $Z^0 \rightarrow < \{AC^2, CC^2\}, \{Inh(CC^2, AC^2)\} >$
1: $AC^0 \rightarrow < \{abstractClass^2 = \{x | \forall x(C(x) \wedge M(x) \wedge Ma(x) \wedge Mt(x))\}\}$, Ø >
2: $CC^0 \rightarrow < \{concreteClass^2 = \{x | \forall x(C(x) \quad Mi(x) \quad \neg Ma(x)\}\}$, Ø >

Dónde:
C(x) = "x es una clase"
Mi(x) = "x tiene métodos implementados"
Ma(x) = "x tiene al menos un método abstracto"
M(x) = "x tiene métodos"
Mt(x) = "x tiene un método TemplateMethod"

R:

1: $Inh(\tau^1, AC^0) \rightarrow [1]\{Inh(\tau^1, abstractClass^2)\}$
$\tau: CC, concreteClass$
2: $Inh(CC^0, \sigma^1) \rightarrow [2]\{Inh(concreteClass^2, \sigma^1)\}$

$\sigma: AC, abstractClass$

5.1.1.3 Proceso que sigue el intérprete

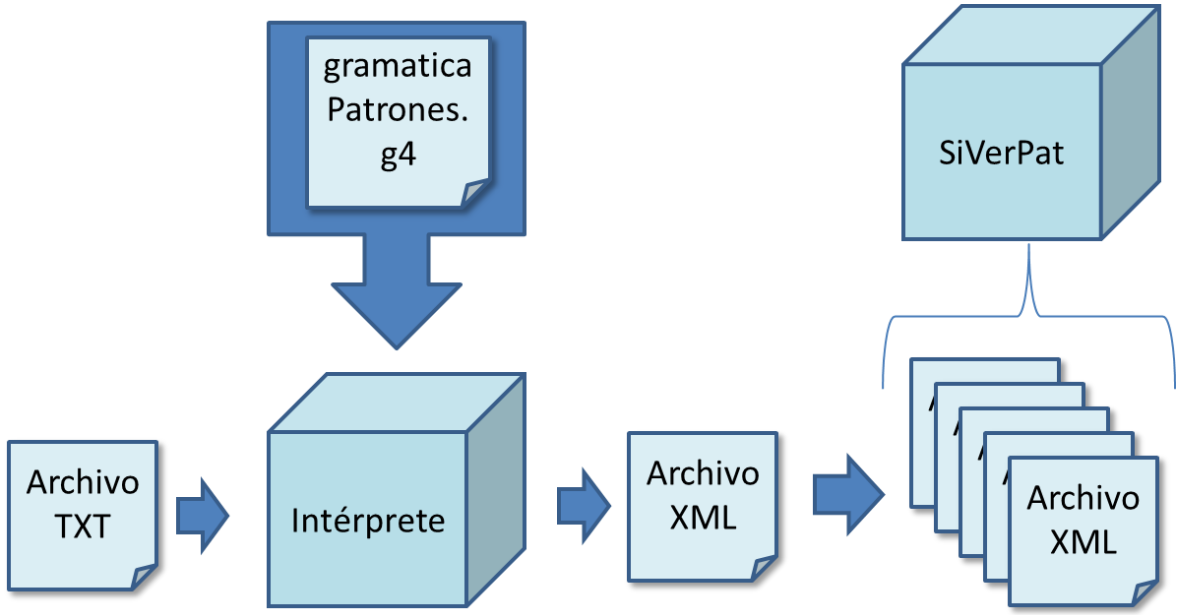

Figura 28. Proceso del intérprete con la herramienta SiVerPat

El archivo debe comenzar con las etiquetas:

```
<?xml version="1.0" encoding="UTF-8"?>
<Rules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Rules.xsd">
```

Ilustración 1. Etiquetas de apertura

La primera parte del archivo, contiene los datos generales del patrón de diseño:

Strategy = (***Nombre del patrón***
{Z, CX, ST, CST, CSST, CSSTX}, ***Símbolos No Terminales***
{context, strategy, concreteStrategy}, ***Símbolos Terminales***
{Inh, Comp}, ***Símbolos de relación***
Z, ***Símbolo inicial***
P, ***Conjunto finito de s-items***
R ***Conjunto finito de r-items***
)

Como primer paso se abre una etiqueta Pattern, dentro de ésta se abre una etiqueta TypePattern con el nombre del patrón de diseño.

```
<Pattern>
  <TypePattern>STRATEGY</TypePattern>
```

Figura 29. Nombre de Strategy en Rules.xml

El siguiente paso consiste en el llenado de los símbolos terminales, este se repite por cada símbolo terminal definido en la gramática, a la etiqueta *Index* se le asigna el valor 1 lo cual indica que es símbolo terminal, la etiqueta *Name* almacena el nombre de ese símbolo terminal y la etiqueta *Role* indica el rol que tomará.

```
<TerminalSymbols>
    <TerminalSymbol>
        <Index>1</Index>
        <Name>Context</Name>
        <Role>CONTEXT</Role>
    </TerminalSymbol>
```

Figura 30. Símbolos terminales en Rules.xml

A continuación se hace el vaciado de la etiqueta S_Productions, que corresponde a la sección P de la gramática. Estas pueden ser terminales (contiene uno de los símbolos terminales previamente definidos)

Entonces la producción terminal

$6: CX^{0} \rightarrow < \{context^{2}\},\ \emptyset >$

Sería llenado de la siguiente forma.

```
<S_Productions>
    <TerminalProduction>
        <Index>6</Index>
        <NonTerminalSymbol>CX</NonTerminalSymbol>
        <TerminalSymbol>Context</TerminalSymbol>
    </TerminalProduction>
```

Figura 31. S_Productions de símbolo terminal en Rules.xml

Existe un caso especial de las producciones terminales, cuando contiene la regla a vacío

$3: CSSTX^{0} \rightarrow < \{\lambda\}, \emptyset >$

Sería llenada de la siguiente forma

```
<TerminalProduction>
    <Index>3</Index>
    <NonTerminalSymbol>CSSTX</NonTerminalSymbol>
    <TerminalSymbol>VoidSymbol</TerminalSymbol>
</TerminalProduction>
```

Figura 32. S_Productions de símbolo terminal vacio en Rules.xml

Las producciones no terminales se identifican porque no contienen un símbolo terminal, entonces la producción no terminal

$1: CSST^{0} \rightarrow < \{CST^{2}\ CSSTX^{2}\},\ \emptyset >$

Sería llenada de la siguiente forma

```
<NonTerminalProduction>
    <Index>1</Index>
    <NonTerminalSymbol>CSST</NonTerminalSymbol>
    <SymbolsContained>CST</SymbolsContained>
    <SymbolsContained>CSSTX</SymbolsContained>
</NonTerminalProduction>

</S_Productions>
```

Figura 33. S_Productions de símbolo no terminal en Rules.xml

Una vez capturadas las S-producciones, se capturan las relaciones terminales del patrón, las cuales están definidas en el símbolo inicial del patrón, normalmente será el elemento Z y su índice será el 0.

$0: Z^0 \rightarrow < \{CX^2, ST^2, CSST^2\}, \{Inh(CSST^2, ST^2), Comp(CX^2, ST^2)\} >$

Estas relaciones se llenan en la etiqueta de la siguiente forma.

```
<TerminalRelationships>
    <TerminalRelationship>
        <TypeRelationship>INH</TypeRelationship>
        <Source>CSST</Source>
        <Destination>ST</Destination>
    </TerminalRelationship>
    <TerminalRelationship>
        <TypeRelationship>COMP</TypeRelationship>
        <Source>CX</Source>
        <Destination>ST</Destination>
    </TerminalRelationship>
</TerminalRelationships>
```

Figura 34. Relaciones Terminales en Rules.xml

El siguiente elemento a capturar es el elemento raíz, que es la producción con el índice 0

$0 Z^0 \rightarrow < \{CX^2, ST^2, CSST^2\}, \{Inh(CSST^2, ST^2), Comp(CX^2, ST^2)\} >$

El cual sería llenado de la siguiente forma.

```
<RootElement>
    <NameRootElement>Z</NameRootElement>
    <BaseSymbols>CX</BaseSymbols>
    <BaseSymbols>ST</BaseSymbols>
    <BaseSymbols>CSST</BaseSymbols>
</RootElement>
```

Figura 35. Elemento Raíz en Rules.xml

El siguiente elemento a llenar es la tabla de análisis, su construcción consta de renglones que representan los símbolos no terminales y las columnas representan los roles y el símbolo especial $.

El rol lo obtenemos de derivar la regla hasta un símbolo terminal, por ejemplo el proceso de derivación de es

Símbolo no terminal: **CSSTX** -> *Símbolo no terminal:* **CSST** -> *Símbolo no terminal:* **CST** -> *Símbolo terminal:* **concreteStrategy** -> *Rol del símbolo terminal:* **CONCRETESTRATEGY**

En el caso especial del SimboloVacio (λ) la derivación lleva al símbolo especial de fin de algoritmo ($).

Símbolo Terminal: λ -> Rol del símbolo terminal: $

Entonces las producciones

2: $CSSTX^0 \rightarrow < \{CSST^2\}, \emptyset >$
3: $CSSTX^0 \rightarrow < \lambda, \emptyset >$

Se representan de la siguiente forma

```
<TableAnalysis>

    <Row>
        <NonTerminalSymbol>CSSTX</NonTerminalSymbol>
        <Cell>
            <Role>CONCRETESTRATEGY</Role>
            <IndexS_Production>2</IndexS_Production>
        </Cell>
        <Cell>
            <Role>$</Role>
            <IndexS_Production>3</IndexS_Production>
        </Cell>
    </Row>
```

Figura 36. Representación de una celda en el elemento Tabla de Análisis en Rules.xml

El siguiente elemento a llenar son los comodines, en la gramática se representan por las letras σ y τ, que pueden tomar cualquier valor definido en la gramática. Se alternan las σ y τ para que sea más fácil de leer, para registrarlo en el XML, cada conjunto de posibles valores que puede tomar el comodín igual es el mismo, por ejemplo en la gramática del patrón de diseño Strategy, el comodín ST, Strategy es el mismo en todos, por lo tanto esta gramática solo tiene dos comodines ST, Strategy y CX, Context.

Los comodines siempre empiezan en 0 y su valor va incrementando en 1, sin embargo pueden tener el nombre que sea, siempre y cuando no haya sido usado.

```
<Wildcards>
    <Wildcard>
        <Index>0</Index>
        <Name>Wildcard 0</Name>
        <Symbols>ST</Symbols>
        <Symbols>Strategy</Symbols>
    </Wildcard>
    <Wildcard>
        <Index>1</Index>
        <Name>Wildcard 1</Name>
        <Symbols>CX</Symbols>
        <Symbols>Context</Symbols>
    </Wildcard>
```

Figura 37. Comodines en Rules.xml

El siguiente elemento a llenar son las R_Productions, que son las producciones R en la gramática.

Se representa en el XML de la siguiente forma.

```
<R_Productions>
    <R_Production>
        <Index>0</Index>
        <NonTerminalRelationship>
            <CompoundRelationship>
                <TypeRelationship>INH</TypeRelationship>
                <Source>CSST</Source>
                <Destination>Wildcard 0</Destination>
            </CompoundRelationship>
            <TerminalRelationship>
                <TypeRelationship>INH</TypeRelationship>
                <Source>CST</Source>
                <Destination>Wildcard 0</Destination>
            </TerminalRelationship>
            <TerminalRelationship>
                <TypeRelationship>INH</TypeRelationship>
                <Source>CSSTX</Source>
                <Destination>Wildcard 0</Destination>
            </TerminalRelationship>
        </NonTerminalRelationship>
    </R_Production>
```

Figura 38. R-Productions en Rules.xml

Por último, se tiene que llenar las etiquetas de tabla de derivaciones, la cual está formada por los índices de la S-producción, que es el número entre corchetes, siendo los renglones, y el tipo de relación, que es la relación en la parte del símbolo no terminal, como columna, por último la celda, que es el índice de la R-producción, esto es útil ya que puede haber varias R-Producciones que usen la regla en la S-producción.

Entonces la siguiente regla
0: Inh$(CSST^0, \tau^1) \rightarrow [1]\{Inh(CST^2, \tau^1), Inh(CSSTX^2, \tau^1)\}$
Se añade de la siguiente forma

```
<TableDerivations>
    <Row>
        <Index>1</Index>
        <Cell>
            <TypeRelationship>INH</TypeRelationship>
            <IndexR_Production>0</IndexR_Production>
        </Cell>
    </Row>
```

Figura 39. Celda de la tabla de derivación en Rules.xml

Finalizado este proceso, se obtiene un archivo XML, con las reglas del patrón de diseño traducidas de forma tal que sea aceptado por la herramienta SiVerPat.

CAPÍTULO 6: INTEGRACIÓN DE LAS REGLAS A SIVERPAT

6.1 Análisis de la herramienta SiVerPat

6.1.1 Arquitectura de la herramienta

La herramienta SiVerPat desarrollada por [Salazar, 2011] implemento el patrón MVC, el cual separa el modelo de datos (lógica de la aplicación), el control de eventos y la vista (desplegados de la información) en diferentes capas. Esto permite agregar nuevos elementos a la interfaz sin afectar los ya existentes.

Sin embargo, ya que la gramática usada fue cambiada al idioma inglés, se cambiaron los archivos de reglas, la estructura de las reglas, y varias clases que usaban las referencias en idioma español. La herramienta sigue funcionando igual que antes con la gramática en inglés, sin que se haya añadido nada nuevo.

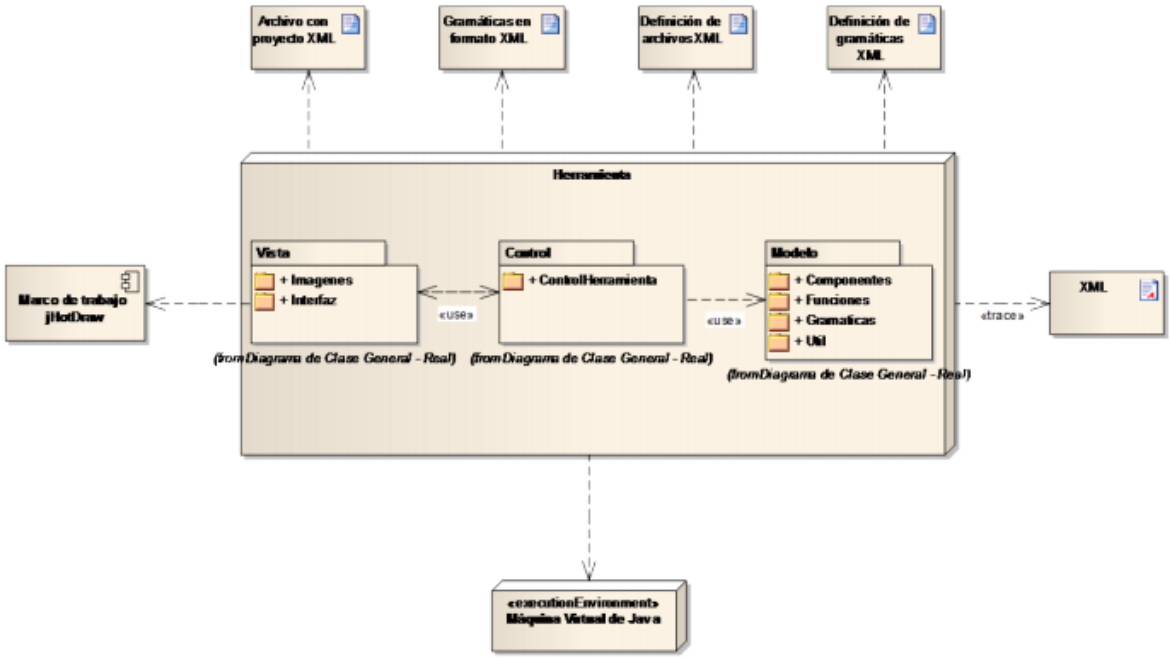

Figura 40. Diagrama de despliegue de la herramienta SiVerPat [Salazar, 2011]

6.1.2 Diagrama de paquetes

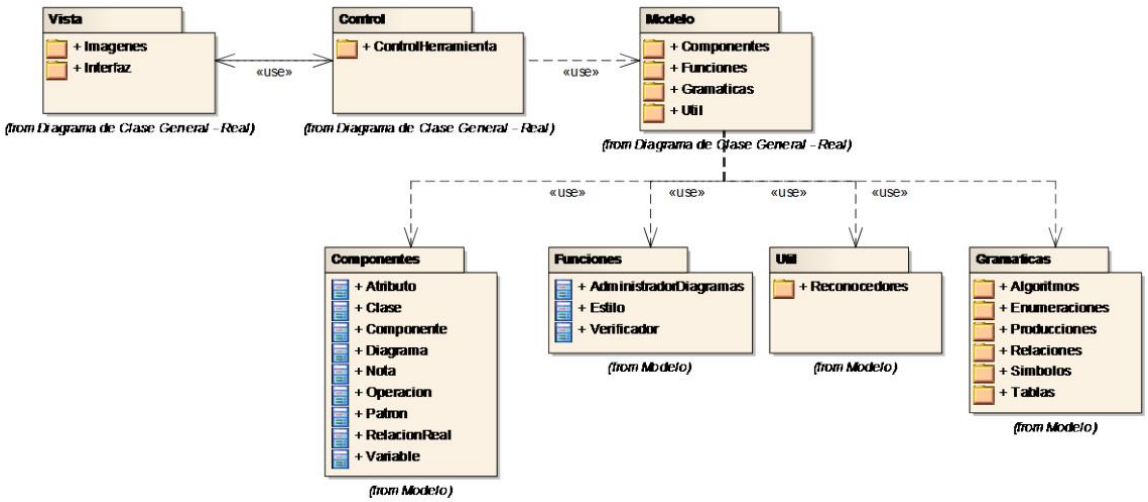

Figura 41. Diagrama de paquetes de la herramienta SiVerPat [Salazar,2011]

La siguiente tabla describe el propósito de cada paquete en la herramienta

Nombre del Paquete	Descripción
Vista	**Contiene la base de la interfaz de la aplicación, consiste en la utilización del marco de aplicación llamado jHotDraw.**
Interfaz	**Contiene a las clases que se dedican a desplegar la información de los diagramas, más no almacenan la información.**
Acciones	**Contiene las acciones ejecutadas al momento de presionar los botones para ver las propiedades de una clase, o agregar patrones al diagrama.**
Figuras	**Contiene las clases que despliegan a la información del diagrama en forma gráfica. Se encuentra dentro del paquete Interfaz**
Herramientas	**Contiene herramientas de creación y selección de patrones.**
Paneles	**Contiene a los paneles que proveen un medio para desplegar, editar o agregar elementos a clases y relaciones.**
Imágenes	**Contiene a los iconos que se muestran en la herramienta.**
Control	**Es el mediador entre la Vista y el Modelo. Recibe las acciones del usuario y las dirige hacia la parte del modelo que se necesite. En viceversa también.**
ControlHerramienta	**Contiene a las clases que controlan la comunicación entre la vista y el modelo.**
Modelo	**Contiene la lógica de la aplicación, desde la creación del diagrama, el reconocimiento de las reglas de combinación, y la verificación de los diagramas.**
Componentes	**Contiene la estructura de clases que representa al diagrama de clases.**
FuncionesGenerales	**Contiene la funcionalidad de administración de diagrama y algunas clases que son necesarias para el sistema como la escritura y lectura de archivos.**
Util	**Contiene los intérpretes de las reglas y los diagramas.**
Gramaticas	**Contiene la funcionalidad que verifica los patrones, así como la estructura necesaria para almacenar las reglas de combinación.**
Algorítmos	**Almacena una serie de algoritmos necesarios para la verificación de patrones.**
Enumeraciones	**Contiene las enumeraciones que se utilizan en el sistema. Contiene tipos de relación, roles, alcances y patrones.**
Producciones	**Contiene la estructura que almacena las reglas que definen a los patrones y sus combinaciones.**

Relaciones	**Contiene la estructura que almacena las relaciones entre elementos.**
Simbolos	**Contiene la estructura que almacena los símbolos, que pueden ser símbolos terminales, no terminales etc. Son utilizados por las reglas.**
Tablas	**Contienen la estructura para generar las tablas donde se guardar las S-Producciones y R-Producciones.**
Resources	**Contiene a las reglas de verificación de patrones en formato XML, así como algunos ejemplos de patrones y combinaciones. No contiene clases.**

Tabla 5. Descripción de paquetes de la herramienta

6.1.3 Cómo funciona la herramienta

Al iniciar la herramienta se nos presenta esta ventana

Figura 42. Ventana principal de SiVerPat

En esta ventana podemos dibujar un nuevo diagrama usando estas utilerías

Figura 43. Botones para la creación del diagrama de SiVerPat

O podemos pulsar alguno de estos botones para crear un patrón de diseño pre-establecido. (En esta imagen ya fue agregado el botón del patrón de diseño Strategy de esta tesis sin interferir con la interfaz previamente hecha)

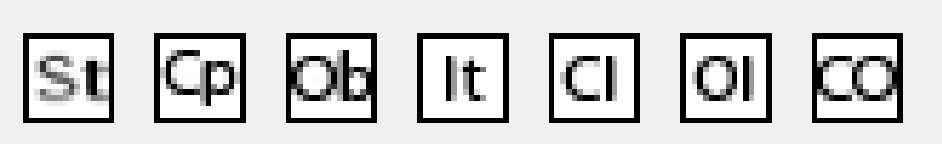

Figura 44. Botones para generar patrones de diseño de SiVerPat

La herramienta creará el diagrama del patrón de diseño que seleccionemos, de acuerdo a como se haya "dibujado" en la clase correspondiente del paquete "Acciones".

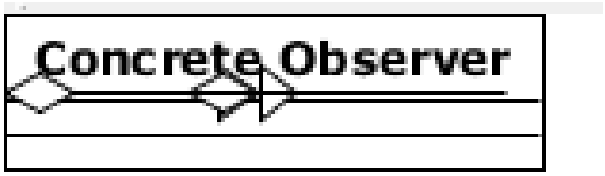

Figura 45. Diagrama Observer creado con SiVerPat

Originalmente el diagrama se muestra "Colapsado", y le corresponde al usuario acomodarlo de la forma en que prefiera.

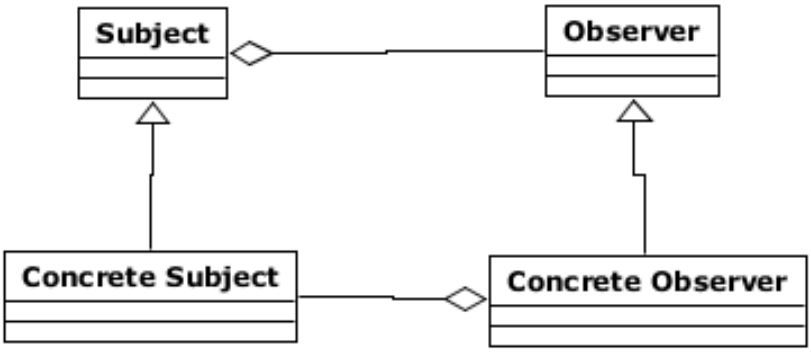

Figura 46. Diagrama Observer acomodado creado con SiVerPat

Con la creación de un patrón de diseño, también se crea una instancia que es verificada, usando el algoritmo LL(1) mostrado anteriormente la cual se le asigna un numero aleatorio.

Figura 47. Mensaje mostrado cuando se verifica el Diagrama Observer

Esta verificación se mantiene constante, si se llegará a realizar un cambio que afecte la integridad del patrón de diseño, por ejemplo eliminar una clase.

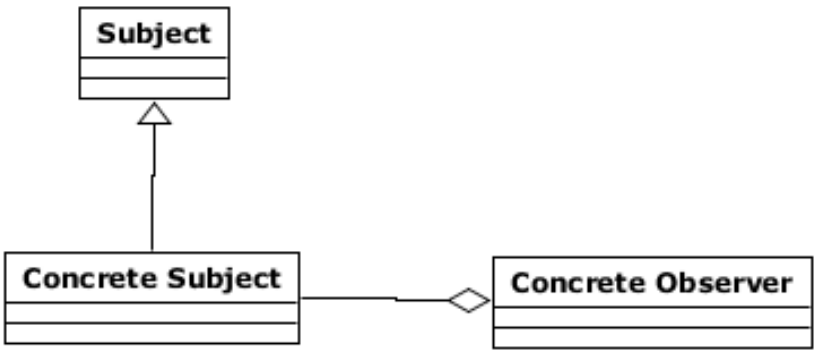

Figura 48. Diagrama Observer sin la clase base "Observer"

La verificación se realizaría presionando este botón.

Figura 49. Botón "Verificar Patrones"

Se volverán a meter los elementos presentes en el diagrama al algoritmo LL(1), ya que no cumple con las reglas establecida, se muestra el mensaje de que no es válido.

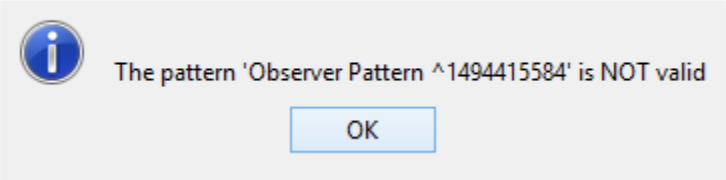

Figura 50. Mensaje del Diagrama Observer no válido.

La herramienta es capaz de guardar y abrir archivos a través del menú archivo.

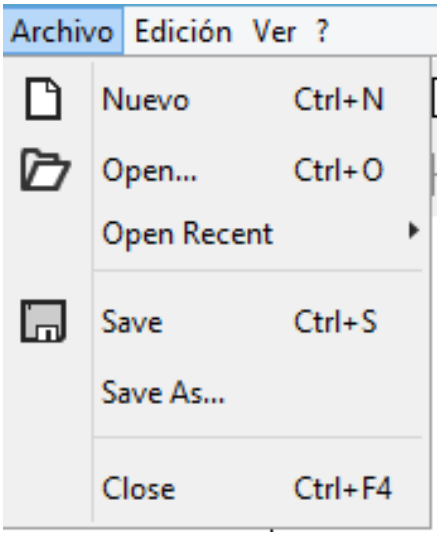

Figura 51. Menú Archivo desglosado.

Nuevo: La opción nuevo crea una nueva ventana del SiVerPat, donde se pueden crear nuevos diagramas (existe el problema que SiVerPat hace todas las acciones en la ventana más nueva, si se interactúa con los botones de una vieja ventana, como por ejemplo pedir verificación se realiza sobre la nueva ventana)
Open: Permite abrir un archivo de formato XML almacenado por SiVerPat. Si cumple con la estructura adecuada, dibujará el diagrama en la ventana.
Save: Guarda el diagrama en un archivo de formato XML.

6.2 Agregar un nuevo patrón o combinación a la herramienta

Para agregar un patrón o alguna nueva combinación a la herramienta es necesario seguir el siguiente proceso. Como ejemplo se muestra el patrón de diseño Strategy y su gramática.

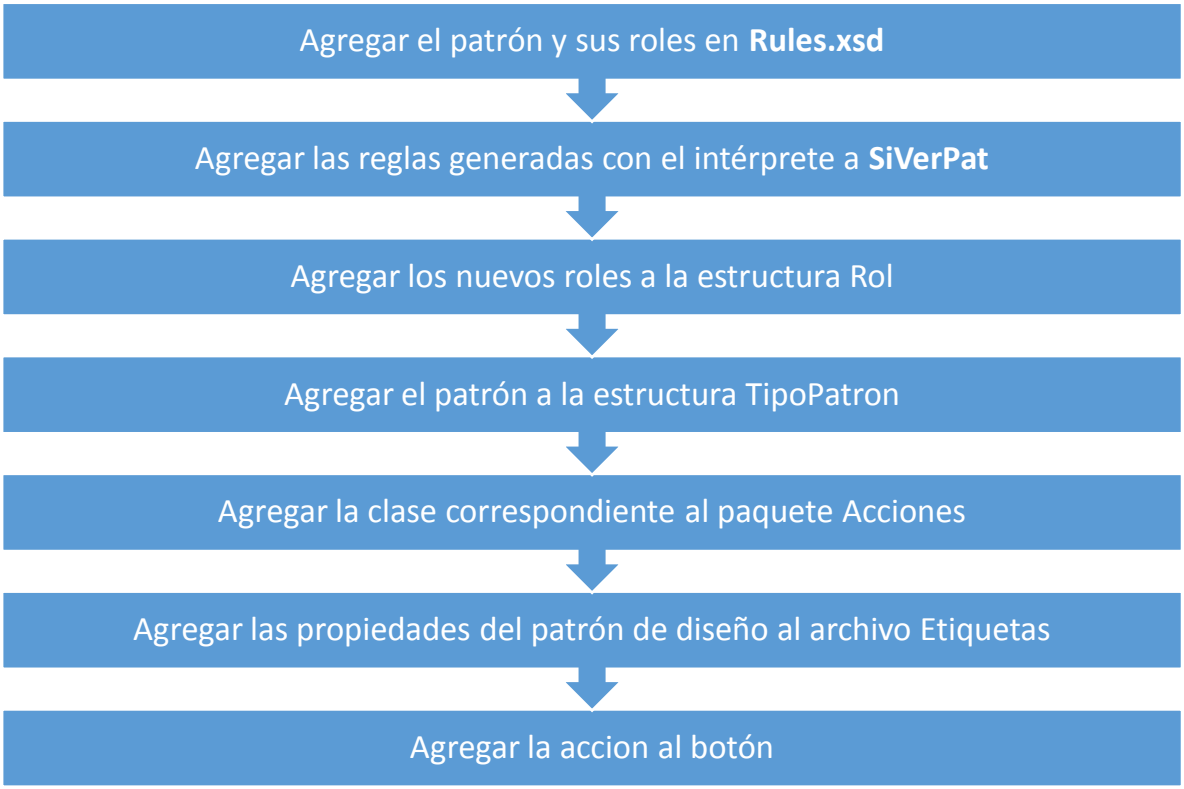

Figura 52. Proceso para agregar un patrón a SiVerPat

6.2.1 Agregar el patrón y sus roles en Rules.xsd

Es necesario agregar el nombre del patrón en el xsd en la sección TypePattern.

```
<xs:simpleType name="TypePattern">
    <xs:restriction base="xs:string">
        <xs:enumeration value="OBSERVER"/>
        <xs:enumeration value="COMPOSITE"/>
        <xs:enumeration value="ITERATOR"/>
        <xs:enumeration value="OBSERVERCOMPOSITE"/>
        <xs:enumeration value="ITERATORX"/>
        <xs:enumeration value="STRATEGY"/>
        <xs:enumeration value="VOID"/>
    </xs:restriction>
</xs:simpleType>
```

Figura 53. TypePattern archivo Rules.xsd

Así como los roles que lo definen en la sección TypeRole.

```
<xs:simpleType name="TypeRole">
    <xs:restriction base="xs:string">
        <xs:enumeration value="SUBJECT"/>
        <xs:enumeration value="CONCRETESUBJECT"/>
        <xs:enumeration value="OBSERVER"/>
        <xs:enumeration value="CONCRETEOBSERVER"/>
        <xs:enumeration value="COMPONENT"/>
        <xs:enumeration value="COMPOSITE"/>
        <xs:enumeration value="LEAF"/>
        <xs:enumeration value="AGGREGATE"/>
        <xs:enumeration value="CONCRETEAGGREGATE"/>
        <xs:enumeration value="ITERATOR"/>
        <xs:enumeration value="CONCRETEITERATOR"/>
        <xs:enumeration value="CONTEXT"/>
        <xs:enumeration value="STRATEGY"/>
        <xs:enumeration value="CONCRETESTRATEGY"/>
        <xs:enumeration value="$"/>
        <xs:enumeration value="VOID"/>
    </xs:restriction>
</xs:simpleType>
```

Figura 54. TypeRole archivo Rules.xsd

6.2.2 Agregar los nuevos roles a la estructura Rol

Es necesario agregar los nuevos roles a la estructura que usa la herramienta, ésta se encuentra en el archivo **Rol.java** del paquete **Modelo / Gramatica / Enumeraciones.**

```
public enum Rol {

    SUBJECT(1),
    CONCRETESUBJECT(2),
    OBSERVER(3),
    CONCRETEOBSERVER(4),
    COMPONENT(5),
    LEAF(6),
    COMPOSITE(7),
    AGGREGATE(8),
    CONCRETEAGGREGATE(9),
    CONCRETEITERATOR(10),
    ITERATOR(11),
    CONTEXT(12),
    STRATEGY(13),
    CONCRETESTRATEGY(14),
    ABSTRACTCLASS(15),
    CONCRETECLASS(16),
    CREATOR(17),
    CONCRETECREATOR(18),
    PRODUCT(19),
    CONCRETEPRODUCT(20),
    $(21),
    VOID(22);
```

Figura 55. Roles que maneja la herramienta, marcado los roles del patrón Strategy

6.2.3 Agregar el patrón a la estructura TipoPatron

También es necesario agregar el nuevo patrón de diseño a la estructura de patrones disponibles.

```
public enum TipoPatron {
    OBSERVER,
    COMPOSITE,
    ITERATOR,
    OBSERVERCOMPOSITE,
    ITERATOROBSERVER,
    ITERATORCOMPOSITE,
    ITERATORX,
    STRATEGY,
    VOID;
}
```

Figura 56. Patrones que maneja la herramienta, marcado el patrón Strategy

6.2.4 Agregar la clase correspondiente al paquete Acciones

Dentro del paquete **Vista / Interfaz / Acciones**, se debe agregar una clase correspondiente al patrón o combinación, de nombre AcccionAgregarPatron<*NombreDelPatron*>.java.

Se declara una variable final y static llamada ID, con el nombre de la configuración que tendrá en el archivo de propiedades.

En esta clase, su constructor llama al editor (clase que dibuja la interfaz) y señala que agregue las propiedades del patrón Strategy.

```
public final static String ID = "edit.agregarStrategy";

public AccionAgregarPatronStrategy(DrawingEditor editor) {
    super(editor);
    ResourceBundleUtil labels = ResourceBundleUtil.getBundle("Vista.Interfaz.Etiquetas");
    labels.configureAction(this, ID);
    updateEnabledState();
}
```

Figura 57. Constructor de la clase AccionAgregarPatronStrategy.java

Además, esta clase contiene la forma en que dibujará el patrón, y al final manda a llamar el verificador.

Ya que la clase es muy grande como para anexarla, sólo se explicarán algunas líneas de código representativas.

Siguiendo la gramática, se dibuja el diagrama usando los métodos aquí mencionados.

Esta línea declara una instancia de "Clase" con nombre "context", con Ámbito público, de nombre "context", le define el rol "Context", y se le declaran las listas ligadas que contendrán sus atributos y sus operaciones, y al final el estilo que tendrá.

```
Clase context = new Clase(Ambito.PUBLIC, "Context", Rol.CONTEXT, new LinkedList<Atributo>(),new
LinkedList<Operacion>(),new Estilo());
```

La siguiente línea declara una instancia de tipo "RelacionReal", llamada "relacion1" de tipo INH(herencia), que unirá las clases "concreteStrategy" como el origen y "strategy" como destino, pertenece al patrón "Strategy", sin darle un nombre.

```
RelacionReal relacion1 = new
RelacionReal(TipoRelacion.INH,concreteStrategy,strategy,TipoPatron.STRATEGY,"");
```

La siguiente línea declara el objeto "figuraContext" de tipo "FiguraClase", a la cual se le inicializa con la variable context, así la figura que se dibujará tendrá acceso a todas las propiedades asignadas.

```
FiguraClase figuraContext = new FiguraClase(context);
```

Las siguientes líneas declaran una figura de herencia de nombre "figuraInherence" la cual se inicializa con "relacion1"y composición de nombre "figuraComposition" que se inicializa con "relacion2".

```
FiguraHerencia figuraInherence = new FiguraHerencia(relacion1);
FiguraAgregacionFuerte figuraComposition = new FiguraAgregacionFuerte(relacion2);
```

A las figuras de las relaciones declaradas se les asignan sus puntos de inicio y final, con el siguiente código. Se manda el punto central del objeto "figuraClase" enviada.

```
figuraInherence.setStartConnector(figuraConcreteStrategy.findConnector(Geom.center(figuraConcreteStr
ategy.getBounds()),figuraInherence));

figuraInherence.setEndConnector(figuraStrategy.findConnector(Geom.center(figuraStrategy.getBounds())
,figuraInherence));
```

Luego se crea una variable de tipo "Patron", a este se le asigna un tipo (esto servirá para tome la regla necesaria para ser verificada), un nombre, y se inicializan sus listas ligadas que contienen las clases, sus notas y sus relaciones.

```
Patron patron = new Patron(TipoPatron.STRATEGY,"Strategy Pattern",new LinkedList<Clase>(), new
LinkedList<Nota>(),new LinkedList<RelacionReal>());
```

Las siguientes líneas, obtienen el "Focus" del diagrama dibujado en la herramienta.

```
Diagrama diagrama = SiVerPat.control.obtenerDiagramaActual();
diagrama.seleccionar();
```

Las siguientes líneas agregan las clases y las relaciones a la variable "patrón".

```
patron.agregarClase(context);
patron.agregarClase(strategy);
patron.agregarClase(concreteStrategy);

patron.agregarRelacion(relacion1);
patron.agregarRelacion(relacion2);
```

Después se agrega la variable patrón al Diagrama actual.

```
diagrama.agregarPatron(patron);
```

Luego, le decimos al editor que dibuje todas las figuras.

```
editor.getActiveView().getDrawing().add(figuraContext);
editor.getActiveView().getDrawing().add(figuraStrategy);

editor.getActiveView().getDrawing().add(figuraConcreteStrategy);
editor.getActiveView().getDrawing().add(figuraInherence);

editor.getActiveView().getDrawing().add(figuraComposition);
```

Se deselecciona el diagrama.

```
diagrama.deseleccionar();
```

Y para finalizar, se mandan a verificar los patrones que están en el diagrama.

```
JOptionPane.showMessageDialog(this.getView().getComponent(), SiVerPat.control.verificarPatrones());
```

6.2.5 Agregar las propiedades del patrón de diseño al archivo Etiquetas

Es necesario agregar las propiedades del botón, su descripción emergente (tooltip) y su imagen.

```
edit.agregarStrategy.text=Add Strategy Pattern
edit.agregarStrategy.toolTipText=Add Strategy Pattern
edit.agregarStrategy.accelerator=
edit.agregarStrategy.mnemonic=
edit.agregarStrategy.icon=${imageDir}/Strategy.png
```

Figura 58. Propiedades de Strategy en Etiquetas.propiedades

6.2.6 Agregar la acción al botón

Para agregar el botón a la interfaz, dentro del paquete **Vista.Interfaz.Panel** en la clase **DrawApplicationModel.java**

```
accion = new AccionAgregarPatronStrategy(editor);
accion.setEnabled(true);
tb.add(accion);
```

Figura 59. Líneas para agregar el botón Strategy a la interfaz

Figura 48. Líneas para agregar el botón Strategy a la interfaz

6.3 Incorporar las reglas a la herramienta SiVerPat

El archivo XML producido el intérprete debe ser incluido en la carpeta **src\resources\archivos\patrones,** todas las reglas son archivos XML.

Al iniciar la herramienta SiVerPat se cargaran estas reglas.

```
"C:\Program ...
Pattern rules are read:COMPOSITE
Pattern rules are read:FACTORYMETHOD
Pattern rules are read:ITERATOR
Pattern rules are read:OBSERVER
Pattern rules are read:OBSERVERCOMPOSITE
Pattern rules are read:STRATEGY
Pattern rules are read:TEMPLATEMETHOD
```

Figura 60. Consola mostrada al correr SiVerPat

CAPÍTULO 7: PRUEBAS

Se realizaron dos actividades de pruebas, Pruebas del intérprete y el plan de pruebas de la herramienta SiVerPat.

7.1 Pruebas del intérprete

Los objetivos de las pruebas del intérprete son:

- Verificar que el intérprete construya los archivos XML de forma correcta.
- Validar que se muestre el mensaje de error en caso de que el archivo TXT tenga errores gramaticales.
- Validar que los archivos XML construidos sean aceptados por la herramienta SiVerPat.

7.1.1 Identificadores

Identificador	Descripcion
PAI_01	Abrir archivo TXT
PAI_02	Guardar XML
PAI_03	Convertir Patrones de la tesis de [Gonz,2010]
PAI_04	Convertir Combinaciones de la tesis de [Gonz,2010]
PAI_05	Convertir Patrones de la tesis [Pérez, 2014] en su forma dinámica
PAI_06	Convertir Patrones de la tesis [Pérez, 2014] en su forma estática
PAI_07	Convertir Combinaciones de la tesis [Pérez, 2014]

Tabla 6. Identificadores de las pruebas

7.1.2 Plan de pruebas

Identificador	**PAI_01**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	28 de Abril del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_01		
Objetivo	Comprobar que el usuario puede cargar un archivo TXT		
Condiciones de ambiente	Iniciar el programa		
Entrada	Ubicación donde está guardado el archivo.		
Resultados esperados	El archivo cargado en las variables del programa		
Procedimiento de la prueba	1. Inicio. 2. El usuario pulsa el botón “Abrir” 3. La herramienta muestra la ventana para seleccionar un archivo 4. El usuario selecciona un archivo. 5. El usuario pulsa el botón “Abrir” 6. La herramienta carga el archivo. 7. Fin		
Excepciones			
Excepción 1	Si la herramienta encuentra un error al abrir el archivo		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	“Abrir” • Nombre del archivo cargado: “strategy.txt”		Se abrió correctamente un proyecto.
Observaciones	El usuario cargo el archivo en el interprete		

Tabla 7. Plan de prueba PAI_01

Identificador	**PAI_02**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	28 de Abril del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_03		
Objetivo	Comprobar que el usuario puede guardar el archivo XML		
Condiciones de ambiente	Se ha concluido la traducción del archivo de forma correcta		
Entrada	Ubicación donde se guardará el archivo.		
Resultados esperados	El archivo guardado como XML		
Procedimiento de la prueba	1. El sistema al terminar de traducir. 2. Muestra la ventana para guardar el archivo 3. El usuario selecciona la carpeta donde se guardará el archivo 4. El usuario le da un nombre al archivo 5. El usuario da clic en “Save” 6. El intérprete guarda el archivo en la dirección especifica 7. Fin		
Excepciones			
Excepción 1	Si la herramienta tiene problemas con permiso de escritura del archivo, debe mostrar una venta de error y se termina el caso de prueba.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	“Save” • Nombre del archivo TXT: “strategy.txt” • Nombre del archivo XML: “strategy.xml”		Se guardó correctamente un proyecto con el nombre que tenía asignado.
Observaciones	El usuario guardo un proyecto		

Tabla 8. Plan de prueba PAI_02

Identificador	**PAI_03**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	28 de Abril del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_02		
Objetivo	Comprobar que el intérprete traduzca de forma correcta las reglas de los patrones que aparecen en la tesis [Gonz,2010]		
Condiciones de ambiente	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Entrada	El archivo TXT cargado para ser traducido.		
Resultados esperados	Las reglas en formato XML para ser guardadas		
Procedimiento de la prueba	1.El usuario pulsa el botón traducir 2. El intérprete usa ANTLR para recorrer las reglas 3. Al recorrer las reglas, se almacenan los valores en variables locales. 4. Se convierten las variables en la estructura XML deseada. 5. FIN		
Excepciones			
Excepción 1	Si el archivo no contiene las reglas según la estructura establecida, mandará un error.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	“Observer” -Original: Las reglas del patrón Observer -Final: El archivo en formato XML de las reglas Observer para ser guardado		Se tradujo el archivo TXT correctamente en formato XML

Observaciones	La herramienta convirtió correctamente las reglas en XML	
No. Instancia	**Valor**	**Resultado esperado**
Instancia 2	"Iterator" -Original: Las reglas del patrón Iterator -Final: El archivo en formato XML de las reglas Iterator para ser guardado	Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML	
No. Instancia	**Valor**	**Resultado esperado**
Instancia 3	"Composite" -Original: Las reglas del patrón Composite -Final: El archivo en formato XML de las reglas Composite para ser guardado	Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML	

Tabla 9. Plan de prueba PAI_03

Identificador	**PAI_04**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	28 de Abril del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_02		
Objetivo	Comprobar que el intérprete traduzca de forma correcta las reglas de las combinaciones de patrones que aparecen en la tesis [Gonz,2010]		
Condiciones de ambiente	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Entrada	El archivo TXT cargado para ser traducido.		
Resultados esperados	Las reglas en formato XML para ser guardadas		
Procedimiento de la prueba	1.El usuario pulsa el botón traducir 2. El intérprete usa ANTLR para recorrer las reglas 3. Al recorrer las reglas, se almacenan los valores en variables locales. 4. Se convierten las variables en la estructura XML deseada. 5. FIN		
Excepciones			
Excepción 1	Si el archivo no contiene las reglas según la estructura establecida, mandará un error.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"ObserverComposite" -Original: Las reglas del patrón ObserverComposite -Final: El archivo en formato XML de las reglas ObserverComposite para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 2	"Iterator" -Original: Las reglas del patrón Iterator -Final: El archivo en formato XML de las reglas Iterator para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	SiVerPat usa la regla de Iterator para las combinaciones de patrones que usan Iterator.		

Tabla 10. Plan de prueba PAI_04

Identificador	PAI_05		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	28 de Abril del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_02		
Objetivo	Comprobar que el intérprete traduzca de forma correcta las reglas de los patrones que aparecen en la tesis [Pérez, 2014] en su forma dinámica.		
Condiciones de ambiente	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Entrada	El archivo TXT cargado para ser traducido.		
Resultados esperados	Las reglas en formato XML para ser guardadas		
Procedimiento de la prueba	1.El usuario pulsa el botón traducir 2. El intérprete usa ANTLR para recorrer las reglas 3. Al recorrer las reglas, se almacenan los valores en variables locales. 4. Se convierten las variables en la estructura XML deseada. 5. FIN		
Excepciones			
Excepción 1	Si el archivo no contiene las reglas según la estructura establecida, mandará un error.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"Strategy" -Original: Las reglas del patrón Strategy en su forma dinámica -Final: El archivo en formato XML de las reglas Strategy para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 2	"Template Method" -Original: Las reglas del patrón Template Method en su forma dinámica -Final: El archivo en formato XML de las reglas Template Method para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 3	"Factory Method" -Original: Las reglas del patrón Factory Method en su forma dinámica -Final: El archivo en formato XML de las reglas Factory Method para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		

Tabla 11. Plan de prueba PAI_05

Identificador	PAI_06		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	28 de Abril del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_02		
Objetivo	Comprobar que el intérprete traduzca de forma correcta las reglas de los patrones que aparecen en la tesis [Pérez, 2014] en su forma estática.		
Condiciones de ambiente	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Entrada	El archivo TXT cargado para ser traducido.		
Resultados esperados	Las reglas en formato XML para ser guardadas		
Procedimiento de la prueba	1.El usuario pulsa el botón traducir 2. El intérprete usa ANTLR para recorrer las reglas 3. Al recorrer las reglas, se almacenan los valores en variables locales. 4. Se convierten las variables en la estructura XML deseada. 5. FIN		
Excepciones			
Excepción 1	Si el archivo no contiene las reglas según la estructura establecida, mandará un error.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"Strategy" -Original: Las reglas del patrón Strategy en su forma estática -Final: El archivo en formato XML de las reglas Strategy para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 2	"Template Method" -Original: Las reglas del patrón Template Method en su forma estática -Final: El archivo en formato XML de las reglas Template Method para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 3	"Factory Method" -Original: Las reglas del patrón Factory Method en su forma estática -Final: El archivo en formato XML de las reglas Factory Method para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		

Tabla 12. Plan de prueba PAI_06

Identificador	PAI_07		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	28 de Abril del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_02		
Objetivo	Comprobar que el intérprete traduzca de forma correcta las reglas de las combinaciones de patrones de diseño que aparecen en la tesis [Pérez, 2014].		
Condiciones de ambiente	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Entrada	El archivo TXT cargado para ser traducido.		
Resultados esperados	Las reglas en formato XML para ser guardadas		
Procedimiento de la prueba	1.El usuario pulsa el botón traducir 2. El intérprete usa ANTLR para recorrer las reglas 3. Al recorrer las reglas, se almacenan los valores en variables locales. 4. Se convierten las variables en la estructura XML deseada. 5. FIN		
Excepciones			
Excepción 1	Si el archivo no contiene las reglas según la estructura establecida, mandará un error.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"Strategy-Composite" -Original: Las reglas de la combinación Strategy-Composite -Final: El archivo en formato XML de la combinación Strategy-Composite para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 2	"Strategy-Template Method" -Original: Las reglas de la combinación Strategy-Template Method -Final: El archivo en formato XML de la combinación Strategy-Template Method para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 3	"Strategy-Factory Method" -Original: Las reglas de la combinación Strategy-Factory Method -Final: El archivo en formato XML de la combinación Strategy-Factory Method para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 4	"Composite- Template Method" -Original: Las reglas de la combinación Composite- Template Method -Final: El archivo en formato XML de la combinación Composite- Template Method para ser guardado		Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML		
No. Instancia	**Valor**		**Resultado esperado**
Instancia 5	"Composite – Factory Method" -Original: Las reglas de la combinación Composite – Factory Method		Se tradujo el archivo TXT correctamente en formato XML

	-Final: El archivo en formato XML de la combinación Composite – Factory Method para ser guardado	
Observaciones	La herramienta convirtió correctamente las reglas en XML	
No. Instancia	**Valor**	**Resultado esperado**
Instancia 6	"Template Method – Factory Method" -Original: Las reglas de la combinación Template Method – Factory Method -Final: El archivo en formato XML de la combinación Template Method – Factory Method para ser guardado	Se tradujo el archivo TXT correctamente en formato XML
Observaciones	La herramienta convirtió correctamente las reglas en XML	

Tabla 13. Plan de prueba PAI_07

7.2 Resultado de las pruebas del intérprete

La siguiente tabla muestra los resultados obtenidos en cada prueba.

Identificador		**Ejecución**
PAI_01	Instancia 1	Aceptada
PAI_02	Instancia 1	Aceptada
PAI_03	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
PAI_04	Instancia 1	Aceptada
	Instancia 2	Aceptada
PAI_05	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
PAI_06	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
PAI_07	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
	Instancia 4	Aceptada
	Instancia 5	Aceptada
	Instancia 6	Aceptada

Tabla 14. Resultado de las pruebas de aceptación del intérprete

En el anexo B se muestra a detalle el resultado de cada instancia de la prueba

7.3 Plan de pruebas

En esta sección se muestran las pruebas que se hicieron al sistema, las pruebas se basan originalmente en las realizadas en (Gonzáles, 2010).

7.3.1 Casos de uso

La herramienta debe cumplir los siguientes casos de uso.

Identificador	Descripción
CU_01	El objetivo del caso de uso es permitir al usuario crear un nuevo proyecto.
CU_02	El objetivo del caso de uso es que el usuario sea capaz de guardar un proyecto nuevo.
CU_03	El objetivo del caso de uso es que el usuario sea capaz de abrir un proyecto previamente creado.
CU_04	El objetivo del caso de uso es que el usuario sea capaz de guardar un proyecto previamente creado.
CU_05	El objetivo del caso de uso es permitir al usuario agregar elementos a un diagrama de clases de UML, dentro de un proyecto creado previamente.
CU_06	El objetivo del caso de uso es permitir al usuario modificar elementos a un diagrama de clases de UML, dentro de un proyecto creado previamente.
CU_07	El objetivo del caso de uso es permitir al usuario eliminar elementos a un diagrama de clases de UML, dentro de un proyecto creado previamente.
CU_08	El objetivo del caso de uso es que el sistema verifique que la estructura de clases se apegue a las reglas que definen a un patrón de diseño.
CU_09	El objetivo del caso de uso es que el sistema verifique que la estructura de clases se apegue a las reglas que definen a una combinación de patrones de diseño.
CU_10	El objetivo del caso de uso es que el sistema sea capaz de crear la estructura básica de un patrón de diseño original (**Composite, Iterator, Observer**)
CU_11	El objetivo del caso de uso es que el sistema sea capaz de crear la combinación de patrones de diseño original (**Composite-Iterator, Composite-Observer, Iterator-Observer**)
CU_12	El objetivo del caso de uso es que el sistema sea capaz de crear la estructura básica de un patrón de diseño agregados (**Composite, Strategy, Template Method, Factory Method**)
CU_13	El objetivo del caso de uso es que el sistema sea capaz de crear la combinación de patrones de diseño agregados (**Composite-Strategy, Composite-Template Method, Composite-Factory Method, Strategy-Factory Method, Strategy-Template Method, Factory Method- Template Method**)

Tabla 15. Casos de uso de la herramienta.

7.3.2 Criterios de Aceptación

Los criterios de aceptación se basan en el cumplimiento exitoso de cada una de las instancias de prueba de cada Caso de Prueba.

La aceptación del sistema se dará si todos los casos, con sus respectivas instancias, son ejecutadas exitosamente obteniendo los resultados esperados.

7.3.3 Criterios de Suspensión

No deberán ser suspendidas las pruebas definidas en este Plan de Pruebas bajo ninguna circunstancias, pues de estas dependen la aceptación final del proyecto propuesto.

7.3.4 Entregables

Los entregables que se tendrán como resultado del diseño y ejecución de las pruebas son los siguientes:

- Documento de Plan de Pruebas que contiene los casos de pruebas
- Reporte de Ejecución de Pruebas, que contiene una bitácora de pruebas y un resumen final de incidentes

7.3.5 Actividades de Prueba

Las actividades que se deben realizar para la preparación y ejecución de las pruebas de aceptación son las siguientes:

- Generación del Documento de Plan de Pruebas
- Diseño de Casos de Prueba
- Preparación del ambiente para las pruebas internas
- Ejecución de Casos de Prueba
- Generación de la Bitácora de Pruebas
- Generación del Resumen final de incidentes.

7.3.6 Condiciones de Ambiente de desarrollo

Java	**Máquina Virtual de Java: JRE 1.8 o superior.**
Sistema Operativo	Windows 7
Herramientas	IntelliJ IDEA 13.1
Hardware	Equipo portátil o computadora de escritorio.

Tabla 16. Condiciones de ambiente de desarrollo

7.3.7 Personal para Ejecución de Pruebas

Los roles involucrados en la ejecución del presente Plan de Pruebas son los siguientes:

- Ejecutor: Responsable de realizar las pruebas aplicando cada una de las instancias, y generar el Reporte de Ejecución de Pruebas.

El ejecutor deberá contar conocimiento de nivel intermedio a avanzado en el lenguaje de programación Java.

7.3.8 Riesgos y Contingencias

Los riesgos y contingencias para la ejecución del plan de pruebas son los siguientes:

- Riesgo: No se dispone del ambiente para la ejecución de pruebas.
- Contingencia: Se pospone la ejecución de las pruebas hasta contar con el ambiente requerido.

- Riesgo: El ejecutor de las pruebas no cumple con el perfil y habilidades requeridas.
- Contingencia: Se pospone la ejecución de las pruebas hasta tener el personal adecuado.

- Riesgo: El sistema no se encuentra construido en tiempo para llevar a cabo las pruebas.
- Contingencia: Se extiende el plazo del tiempo de desarrollo

7.4. Casos de Prueba

A continuación se describen los Casos de Prueba, se definen las características principales, los cursos alternos, excepciones y resultados esperados.

7.4.1 Prueba de aceptación PA_01

Identificador	**PA_01**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_01		
Objetivo	Comprobar que el usuario puede crear un nuevo proyecto		
Condiciones de ambiente	Se ha iniciado la aplicación.		
Entrada	Ninuna		
Resultados esperados	Un nuevo proyecto ha sido creado		
Procedimiento de la prueba	1. Inicio. 2. El usuario pulsa el menú “File”. 3. El usuario selecciona la opción “New”. 4. El sistema debe mostrar una nueva ventana de la herramienta vacía. 5. Fin.		
Cursos alternos			
Curso Alterno 1	Ninguno		
Excepciones			
Excepción 1	El sistema no crea una ventana nueva, debe mostrar un venta de error		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**

Instancia 1	Nueva ventana sin nombre	Una nueva ventana
Observaciones	Se muestra una nueva ventana	

Tabla 17. Prueba de aceptación PA_01

7.4.2 Prueba de aceptación PA_02

Identificador	**PA_02**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_02		
Objetivo	Comprobar que el usuario puede guardar un proyecto nuevo.		
Condiciones de ambiente	Se ha iniciado la aplicación.		
Entrada	Nombre del proyecto y ubicación donde se desea guardar.		
Resultados esperados	Un archivo XML que contiene los elementos del diagrama almacenado.		
Procedimiento de la prueba	1. Inicio. 2. El usuario pulsa el menú "File". 3. El usuario selecciona la opción "Save". 4. Se muestra una ventana donde el usuario seleccionara la carpeta donde se guardara el proyecto. 5. El usuario le da un nombre al proyecto. 6. El usuario pulsa el botón "Save". 7. La herramienta comprobará que no exista un archivo del mismo nombre. 8. La herramienta guardará el archivo. 9. Fin.		
Cursos alternos			
Curso Alterno 1	3.1 El usuario selecciona la opción "Save as". 4.1 Volver al paso 4.		
Curso Alterno 2	El usuario elige un nombre ya existente: 7.2 La herramienta muestra un mensaje de error diciendo que ese nombre ya existe, si desea sobrescribir. 8.2 El usuario pulsa "Ok" para quitar el mensaje 10.2 Volver al paso 8		
Curso Alterno 3	8.2.3 El usuario pulsa "Cancelar" 8.2.4 El usuario cambia el nombre del archivo 8.2.5 Volver al paso 6		
Curso Alterno 4	El usuario pulsa "Cancelar" 7.3 La herramienta no guarda ningún archivo 8.3 Fin		
Excepciones			
Excepción 1	Si la herramienta tiene problemas con permiso de escritura del archivo, debe mostrar una venta de error y se termina el caso de prueba.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"Save" • Nombre del proyecto original: "Proyecto01" • Nombre del proyecto final: "Proyecto 01"		Se guardó correctamente un proyecto con el nombre "Proyecto01"
Observaciones	El usuario guardo un proyecto		
Instancia 2	"Save as" • Nombre del proyecto original: "Proyecto02" • Nombre del proyecto final: "Proyecto 02"		Se guardó correctamente un proyecto con el nombre "Proyecto02"

Observaciones	Se utiliza el curso alterno 1	
Instancia 3	"Mismo nombre" • Nombre del proyecto original: "Proyecto01" • Nombre del proyecto final: Ninguno	Se muestra un mensaje de error de "El archivo ya existe"
Observaciones	Se utilizó el curso alterno 2	
Instancia 4	"Cancelar el sobrescribir" • Nombre del proyecto original: "Proyecto01" • Nombre del proyecto final: Ninguno	Se debe mostrar la ventana de guardar para que el usuario cambie el nombre.
Observaciones	Se utiliza el curso alterno 3	
Instancia 5	"Cancelar" • Nombre del proyecto original: "Proyecto01" Nombre del proyecto final: Ninguno	No se debe de crear ningún archivo, ya que se canceló la acción
Observaciones	Se utilizó el curso alterno 4	

Tabla 18. Prueba de aceptación PA_02

7.4.3 Prueba de aceptación PA_03

Identificador	**PA_03**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_03		
Objetivo	Comprobar que el usuario puede abrir un proyecto nuevo.		
Condiciones de ambiente	Los casos de prueba PA_01 y PA_02 se ejecutaron con éxito		
Entrada	Nombre del proyecto y ubicación donde se desea abrir el archivo.		
Resultados esperados	Mostrar el diagrama contenido en el archivo.		
Procedimiento de la prueba	1. Inicio. 2. El usuario pulsa el menú "File". 3. El usuario selecciona la opción "Open". 4. Se muestra una ventana donde el usuario seleccionara la carpeta donde está guardado el proyecto. 5. El usuario selecciona el proyecto. 6. El usuario pulsa el botón "Open". 7. La herramienta verifica que el archivo sea válido. 8. La herramienta dibuja el diagrama en la ventana. 9. Fin.		
Cursos alternos			
Curso Alterno 1	El usuario pulsa "Cancelar" 6.1 La herramienta no abre ningún archivo 7.1 Fin		
Curso Alterno 2	El usuario elige un archivo no valido: 8.2 La herramienta muestra un mensaje de error diciendo que el archivo no es válido. 9.2 Fin		
Excepciones			
Excepción 1	Si la herramienta tiene problemas abriendo el archivo, debe mostrar una venta de error y se termina el caso de prueba.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"Open"		Se abrió correctamente un

	• Nombre del proyecto original: "Proyecto01.xml"	proyecto correctamente
Observaciones	El usuario abrió un proyecto	
Instancia 2	"Cancelar" • Nombre del proyecto : "Proyecto01.xml"	No se debe de abrir ningún archivo, ya que se canceló la acción
Observaciones	Se utiliza el curso alterno 1	
	"Archivo no valido" • Nombre del proyecto: "Proyecto01.xml"	Se muestra un mensaje de error de "El archivo no valido"
Observaciones	Se utilizó el curso alterno 2	

Tabla 19. Prueba de aceptación PA_03

7.4.4 Prueba de aceptación PA_04

Identificador	**PA_04**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_04		
Objetivo	Comprobar que el usuario puede guardar un proyecto previamente creado.		
Condiciones de ambiente	Los casos de pruebas PA_01, PA_02 y PA_03 se ejecutaron con éxito.		
Entrada	Nombre del proyecto y ubicación donde está guardado el archivo.		
Resultados esperados	Un archivo XML que contiene los elementos del diagrama almacenado.		
Procedimiento de la prueba	1. Inicio. 2. El usuario pulsa el menú "File". 3. El usuario selecciona la opción "Save". 4. La herramienta guarda el archivo en la misma ubicación. 5. Fin		
Cursos alternos			
Curso Alterno 1	3.1 El usuario selecciona la opción "Save as". 4.1 Se cambia al Caso de prueba PA_02 en el paso 4.		
Excepciones			
Excepción 1	Si la herramienta tiene problemas con permiso de escritura del archivo, debe mostrar una venta de error y se termina el caso de prueba.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"Save" • Nombre del proyecto original: "Proyecto01" • Nombre del proyecto final: "Proyecto 01"		Se guardó correctamente un proyecto con el nombre que tenía asignado.
Observaciones	El usuario guardo un proyecto		

Tabla 20. Prueba de aceptación PA_04

7.4.5 Prueba de aceptación PA_05

Identificador	PA_05		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_05		
Objetivo	Comprobar que el usuario es capaz de agregar elementos a un diagrama de clases de UML, dentro de un proyecto creado previamente.		
Condiciones de ambiente	El caso de prueba PA_01 se ejecutó con éxito.		
Entrada	Información del elemento a ser agregado.		
Resultados esperados	Se agregaron correctamente elementos al diagrama UML		
Procedimiento de la prueba	1. Inicio. 2. El usuario abre un proyecto existente o crea un nuevo proyecto. 3. El usuario selecciona la opción de "Agregar Clase". 4. El usuario selecciona la posición en la que aparecerá la clase. 5. La herramienta dibujará el elemento en el diagrama. 6. Fin.		
Cursos alternos			
Curso Alterno 1	Agregar Herencia 3.1.1 El usuario selección la opción "Crear relación de herencia" 3.1.2 El usuario posiciona el mouse sobe la clase Origen 3.1.3 El usuario arrastra el mouse hasta estar sobre la clase Destino 3.1.4 El usuario libera el mouse. 3.1.5 La herramienta dibuja la relación 3.1.6 Fin		
Curso Alterno 2	Agregar Instanciación 3.2.1 El usuario selección la opción "Crear relación de instanciación" 3.2.2 El usuario posiciona el mouse sobe la clase Origen 3.2.3 El usuario arrastra el mouse hasta estar sobre la clase Destino 3.2.3 El usuario libera el mouse. 3.2.4 La herramienta dibuja la relación 3.2.5 Fin		
Curso Alterno 3	Agregar Agregación 3.3.1 El usuario selección la opción "Crear relación de agregación" 3.3.2 El usuario posiciona el mouse sobe la clase Origen 3.3.3 El usuario arrastra el mouse hasta estar sobre la clase Destino 3.3.4 El usuario libera el mouse. 3.3.5 La herramienta dibuja la relación 3.3.6 Fin		
Curso Alterno 4	Agregar Composición 3.4.1 El usuario selección la opción "Crear relación de composición" 3.4.2 El usuario posiciona el mouse sobe la clase Origen 3.4.3 El usuario arrastra el mouse hasta estar sobre la clase Destino 3.4.4 El usuario libera el mouse. 3.4.5 La herramienta dibuja la relación 3.4.6 Fin		
Curso Alterno 5	Agregar Relación a notas 3.5.1 El usuario selección la opción "Crear relación para notas" 3.5.2 El usuario posiciona el mouse sobe la clase Origen 3.5.3 El usuario arrastra el mouse hasta estar sobre la clase Destino 3.5.4 El usuario libera el mouse. 3.5.5 La herramienta dibuja la relación 3.5.6 Fin		

Curso Alterno 6	Agregar Nota 3.6.1 El usuario selecciona la opción de "Agregar Nota". 3.6.2 El usuario selecciona la posición en la que aparecerá la clase. 3.6.2 La herramienta dibujará el elemento en el diagrama. 3.6.3 Fin.	
Curso Alterno 7	El usuario no elige elemento final de una relación 3.7.1 El usuario NO elige un elemento destino 3.7.2 La herramienta no dibuja la relación. 3.7.3 Fin	
Excepciones		
Excepción 1	Si la herramienta tiene problemas creando algún elemento, se debe mostrar una ventana de error indicándolo.	
Instancias de Prueba		
No. Instancia	**Valor**	**Resultado esperado**
Instancia 1	"Agregar Clase"	Se dibuja correctamente una clase en el diagrama
Observaciones	El usuario dibujo una clase	
Instancia 2	"Agregar Herencia"	Se dibuja correctamente una relación de herencia
Observaciones	Se utiliza el curso alterno 1	
Instancia 3	"Agregar Agregación"	Se dibujó correctamente una relación de agregación
Observaciones	Se utilizó el curso alterno 2	
Instancia 4	"Agregar Composición"	Se dibujó correctamente una relación de composición
Observaciones	Se utilizó el curso alterno 3	
Instancia 5	"Agregar Relación vacía"	Se dibujó correctamente una relación vacía
Observaciones	Se utilizó el curso alterno 4	
Instancia 6	"Agregar Nota"	Se dibujó correctamente una nota
Observaciones	Se utilizó el curso alterno 5	
Instancia 7	"No se eligió elemento destino"	No se debe dibujar ningún tipo de relación.
Observaciones	Se utilizó el curso alterno 6	

Tabla 21. Prueba de aceptación PA_05

7.4.6 Prueba de aceptación PA_06

Identificador	**PA_06**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_06		
Objetivo	Comprobar que el usuario es capaz de modificar elementos de un diagrama de clases de UML, dentro de un proyecto creado previamente.		
Condiciones de ambiente	Los casos de prueba PA_01 y PA_05 se ejecutaron con éxito.		
Entrada	Elemento que será modificado		
Resultados esperados	Se modifica correctamente el elemento seleccionado del diagrama UML		
Procedimiento de la prueba	1. Inicio. 2. El usuario abre un proyecto existente o crea un nuevo proyecto, el cual contiene un diagrama con clases y relaciones entre ellas. 3. El usuario da doble clic sobre el elemento que desea modificar.		

	4. Se abre una ventana donde se muestran las propiedades del elemento. 5. El usuario modifica los elementos deseados. 6. El usuario da clic en "Ok". 7. La herramienta guarda los cambios del elemento. 8. Fin.	
Cursos alternos		
Curso Alterno 1	El usuario da clic en Cancelar 6.1 El usuario selección la opción "Crear relación de herencia" 6.2 La herramienta no hace ningún tipo de cambio al elemento seleccionado. 6.3 Fin	
Excepciones		
Excepción 1	Si la herramienta tiene problemas a guardar los cambios, se debe mostrar una ventana de error indicándolo.	
Instancias de Prueba		
No. Instancia	**Valor**	**Resultado esperado**
Instancia 1	"Modificar Clase"	Se muestra la ventana de propiedades de clase, y se realizan los cambios.
Observaciones	El usuario modifica una clase	
Instancia 2	"Modificar Herencia"	Se muestra la ventana de propiedades de herencia, y se realizan los cambios.
Observaciones	El usuario modifica una relación de herencia	
Instancia 3	"Modificar Agregación"	Se muestra la ventana de propiedades de agregación, y se realizan los cambios.
Observaciones	El usuario modifica una relación de agregación	
Instancia 4	"Modificar Composición"	Se muestra la ventana de propiedades de composición, y se realizan los cambios.
Observaciones	El usuario modifica una relación de composición	
Instancia 5	"Modificar Relación para nota"	Se muestra la ventana de propiedades de una relación vacía, y se realizan los cambios.
Observaciones	El usuario modifica una relación para nota	
Instancia 6	"Modificar Nota"	Se muestra la ventana de propiedades de una nota, y se realizan los cambios.
Observaciones	El usuario modifica una nota	
Instancia 7	"Cancelar"	No se debe dibujar ningún tipo de cambio en el elemento.
Observaciones	Se utilizó el curso alterno 1	

Tabla 22. Prueba de aceptación PA_06

7.4.7 Prueba de aceptación PA_07

Identificador	**PA_07**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_07		
Objetivo	Comprobar que el usuario es capaz de eliminar elementos de un diagrama de clases de UML, dentro de un proyecto creado previamente.		
Condiciones de ambiente	Los casos de prueba PA_01 y PA_05 se ejecutaron con éxito.		
Entrada	Elemento que será eliminado		
Resultados esperados	Se elimina correctamente el elemento seleccionado del diagrama UML		
Procedimiento de la prueba	1. Inicio. 2. El usuario abre un proyecto existente o crea un nuevo proyecto, el cual contiene un diagrama con clases y relaciones entre ellas. 3. El usuario selecciona el elemento que desea eliminar haciendo clic sobre él. 4. Presiona el botón "DEL" o "SUPR" para eliminarlo. 6. La herramienta elimina el elemento del diagrama. 7. Fin.		
Cursos alternos			
Curso Alterno 1	Ninguno		
Excepciones			
Excepción 1	Si la herramienta tiene problemas a eliminar un elemento, se debe mostrar una venta de error indicándolo.		
Instancias de Prueba			
No. Instancia	**Valor**	**Resultado esperado**	
Instancia 1	"Eliminar Clase"	La clase selecciona es eliminada, junto con todas su relaciones.	
Observaciones	El usuario elimina una clase		
Instancia 2	"Eliminar Herencia"	La relación de herencia es eliminada.	
Observaciones	El usuario elimina una relación de herencia		
Instancia 3	"Eliminar Agregación"	La relación de agregación es eliminada.	
Observaciones	El usuario elimina una relación de agregación		
Instancia 4	"Eliminar Composición"	La relación de composición es eliminada.	
Observaciones	El usuario elimina una relación de composición		
Instancia 5	"Eliminar Relación para nota"	La relación vacía es eliminada.	
Observaciones	El usuario elimina una relación para nota		
Instancia 6	"Eliminar Nota"	Se elimina la nota con todas las relaciones vacías que tenga.	
Observaciones	El usuario elimina una nota		

Tabla 23. Prueba de aceptación PA_07

7.4.8 Prueba de aceptación PA_08

Identificador	PA_08		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_08		
Objetivo	Comprobar que la herramienta es capaz de verificar patrones de diseño en el diagrama.		
Condiciones de ambiente	Los casos de prueba PA_01, PA_10, PA_11, PA_12 y PA_13se ejecutaron con éxito.		
Entrada	Diagrama de clases que contiene al menos un patrón de diseño, elaborado previamente utilizando la herramienta.		
Resultados esperados	La herramienta mandará un mensaje en donde notificará si el patrón de diseño es válido.		
Procedimiento de la prueba	1. Inicio. 2. El usuario abre un proyecto existente o crea un nuevo proyecto, el cual contiene un diagrama con clases y relaciones entre ellas. 3. El usuario presiona el botón "Verificar Patrones". 4. La herramienta toma los elementos del diagrama, verifica cuales elementos están asociado a cual combinación o patrón de diseño, para usar las reglas de ese patrón de diseño o combinación. 6. La herramienta manda un mensaje con el resultado de la verificación. 7. Fin.		
Cursos alternos			
Curso Alterno 1	Hay elementos sin asociar en el diagrama. 5.1 La herramienta emite un mensaje con el resultado de la verificación, y una lista de los elementos que no están asociados a alguna regla de patrón de diseño o combinación de patrones de diseño.		
Curso Alterno 2	Verificación automática. 3.1 El usuario realiza una acción de edición sobre el diagrama (Agregar una relación, modificar una clase, eliminar una clase o relación) 3.2 Volver al paso 4		
Excepciones			
Excepción 1	Si al solicitar una verificación no hay elemento en el diagrama. Se manda un mensaje notificando esto.		
Instancias de Prueba			
No. Instancia	**Valor**	**Resultado esperado**	
Instancia 1	"Patrón válido"	Un mensaje notificando que el patrón es válido.	
Observaciones	El usuario verifica un patrón que resulta valido		
Instancia 2	"Patrón No válido"	Un mensaje notificando que el patrón no es válido.	
Observaciones	El usuario verifica un patrón que resulta no valido		

Tabla 24. Prueba de aceptación PA_08

7.4.9 Prueba de aceptación PA_09

Identificador	PA_09		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_09		
Objetivo	Comprobar que la herramienta es capaz de verificar combinación de patrones de diseño en el diagrama.		
Condiciones de ambiente	Los casos de prueba PA_01, PA_10, PA_11, PA_12 y PA_13 se ejecutaron con éxito.		
Entrada	Diagrama de clases que contiene al menos una combinación de patrones de diseño, elaborado previamente utilizando la herramienta.		
Resultados esperados	La herramienta mandará un mensaje en donde notificará si la combinación de patrones de diseño es válido.		
Procedimiento de la prueba	1. Inicio. 2. El usuario abre un proyecto existente o crea un nuevo proyecto, el cual contiene un diagrama con clases y relaciones entre ellas. 3. El usuario presiona el botón "Verificar Patrones". 4. La herramienta toma los elementos del diagrama, verifica cuales elementos están asociado a cual combinación o patrón de diseño, para usar las reglas de ese patrón de diseño o combinación. 5. La herramienta emite un mensaje con el resultado de la verificación. 6. Fin.		
Cursos alternos			
Curso Alterno 1	Hay elementos sin asociar en el diagrama. 5.1 La herramienta emite un mensaje con el resultado de la verificación, y una lista de los elementos que no están asociados a alguna regla de patrón de diseño o combinación de patrones de diseño.		
Curso Alterno 2	Verificación realizada por la herramienta. 3.1 El usuario realiza una acción de edición sobre el diagrama (Agregar una relación, modificar una clase, eliminar una clase o relación) 3.2 Volver al paso 4		
Excepciones			
Excepción 1	Si se solicita la verificación y no hay elemento en el diagrama. Se emite un mensaje notificando esto.		
Instancias de Prueba			
No. Instancia	**Valor**		**Resultado esperado**
Instancia 1	"Combinación válida"		Un mensaje notificando que la combinación de patrones es válido.
Observaciones	El usuario verifica una combinación de patrones que resulta valido		
Instancia 2	"Combinación No válida"		Un mensaje notificando que la combinación de patrones no es válido.
Observaciones	El usuario verifica una combinación de patrones que resulta no valido		

Tabla 25. Prueba de aceptación PA_09

7.4.10 Prueba de aceptación PA_10

Identificador	PA_10		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_10 y CU_12		
Objetivo	Comprobar que la herramienta es capaz de crear la estructura básica de un patrón seleccionado.		
Condiciones de ambiente	El caso de prueba PA_01 fue ejecutado con éxito.		
Entrada	Se selecciona el patrón que se desea dibujar en el diagrama de clases.		
Resultados esperados	La herramienta dibujará en el diagrama de clases del patrón de diseño seleccionado.		
Procedimiento de la prueba	1. Inicio. 2. El usuario abre un proyecto existente o crea un nuevo proyecto. 3. El usuario presiona el botón del patrón deseado. 4. La herramienta dibujará en el diagrama de clases el patrón de diseño seleccionado. 7. Fin.		
Cursos alternos			
Curso Alterno 1	Ninguno		
Excepciones			
Excepción 1	Ninguna		
Instancias de Prueba			
No. Instancia	**Valor**	**Resultado esperado**	
Instancia 1	"Crear Composite"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Composite.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Composite.		
Instancia 2	"Crear Iterator"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Iterator.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Iterator.		
Instancia 3	"Crear Observer"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Observer.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Observer.		
Instancia 4	"Crear Strategy"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Strategy.		
Instancia 5	"Crear Template Method"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Template Method.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Template Method.		
Instancia 6	"Crear Factory Method"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Factory Method.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Factory Method.		

Tabla 26. Prueba de aceptación PA_10

7.4.11 Prueba de aceptación PA_11

Identificador	**PA_11**		
Autor	Manuel Bernardo Ibáñez Ocampo	**Fecha**	5 de Diciembre del 2014
Tipo de prueba	Prueba de Aceptación		
Caso de uso	CU_11 y CU_13		
Objetivo	Comprobar que la herramienta es capaz de crear la estructura básica de un patrón seleccionado.		
Condiciones de ambiente	El caso de prueba PA_01 fue ejecutado con éxito.		
Entrada	Se selecciona la combinación de patrones que se desea dibujar en el diagrama de clases.		
Resultados esperados	La herramienta dibujará en el diagrama de clases la combinación de patrones de diseño seleccionado.		
Procedimiento de la prueba	1. Inicio. 2. El usuario abre un proyecto existente o crea un nuevo proyecto. 3. El usuario presiona el botón de la combinación de patrones deseado. 4. La herramienta dibujará en el diagrama de clases la combinación de patrones de diseño seleccionada. 7. Fin.		
Cursos alternos			
Curso Alterno 1	Ninguno		
Excepciones			
Excepción 1	Ninguna		
Instancias de Prueba			
No. Instancia	**Valor**	**Resultado esperado**	
Instancia 1	“Crear Composite-Iterator”	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Composite-Iterator.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Composite-Iterator.		
Instancia 2	“Crear Iterator-Observer”	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Iterator-Observer.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Iterator-Observer.		
Instancia 3	“Crear Observer-Composite”	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Observer-Composite.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Observer-Composite.		
Instancia 4	“Crear Strategy-Composite”	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy-Composite.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Strategy-Composite.		
Instancia 5	“Crear Composite-Template Method”	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Composite-Template Method.	
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Composite-Template Method.		
Instancia 6	“Crear Composite-Factory Method”	Se dibuja en el área de trabajo la	

		arquitectura básica de clases para el patrón de diseño Composite-Factory Method.
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Composite-Factory Method.	
Instancia 7	"Crear Strategy-Template Method"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy-Template Method.
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Strategy-Template Method.	
Instancia 8	"Crear Strategy-Factory Method"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy-Factory Method.
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Strategy-Factory Method.	
Instancia 9	"Crear Template Method-Factory Method"	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Template Method-Factory Method.
Observaciones	Se dibuja la arquitectura básica de clases para el patrón Template Method-Factory Method.	

Tabla 27. Prueba de aceptación PA_11

7.5 Resultado del plan de pruebas de SiVerPat

En la siguiente tabla se muestran los resultados de las prueba de aceptación del plan de pruebas de la tesis.

Identificador		**Ejecución**
PA_01	Instancia 1	Aceptada
PA_02	Instancia 1	Aceptada
	Instancia 2	Rechazada
	Instancia 3	Rechazada
	Instancia 4	Rechazada
	Instancia 5	Aceptada
PA_03	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Rechazada
PA_04	Instancia 1	Aceptada
PA_05	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
	Instancia 4	Aceptada
	Instancia 5	Aceptada
	Instancia 6	Aceptada
	Instancia 7	Aceptada
PA_06	Instancia 1	Rechazada
	Instancia 2	Rechazada
	Instancia 3	Rechazada
	Instancia 4	Rechazada
	Instancia 5	Rechazada

	Instancia 6	Rechazada
	Instancia 7	Aceptada
PA_07	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
	Instancia 4	Aceptada
	Instancia 5	Aceptada
	Instancia 6	Aceptada
PA_08	Instancia 1	Aceptada
	Instancia 2	Aceptada
PA_09	Instancia 1	Aceptada
	Instancia 2	Aceptada
PA_10	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
	Instancia 4	Aceptada
	Instancia 5	Aceptada
	Instancia 6	Aceptada
PA_11	Instancia 1	Aceptada
	Instancia 2	Aceptada
	Instancia 3	Aceptada
	Instancia 4	Aceptada
	Instancia 5	Aceptada
	Instancia 6	Aceptada
	Instancia 7	Aceptada
	Instancia 8	Aceptada
	Instancia 9	Aceptada

Tabla 28. Tabla de resultados de plan de pruebas

En el anexo C se muestra a detalle el resultado del plan de prueba de las tesis.

CAPÍTULO 8: CONCLUSIONES

En esta investigación se desarrolló un intérprete complementario a *SiVerPat* el cual permite agregar nuevas reglas de patrones de diseño o combinaciones directamente a *SiVerPat* facilitando al usuario agregar nuevas reglas siempre y cuando estas reglas sean gramáticamente correctas.

Con el funcionamiento del intérprete se han cumplido los objetivos de la tesis debido a que con su uso se agregan con éxito las reglas de los patrones *Strategy*, *Template Method*, *Factory Method* y sus combinaciones entre estos y el patrón *Composite*. También se han cumplido los objetivos de agregar a *SiVerPat* la capacidad de dibujar estos diagramas y combinaciones en su forma estática. Además se modificó *SiVerPat* para poder cargar archivos XML de reglas de forma independiente lo que resuelve el problema de tener un solo archivo monolítico, si se llegará a presentar un error con reglas nuevas pueden ser modificadas de forma independiente sin afectar las reglas que funcionan de forma correcta.

SiVerPat dibuja correctamente los diagramas de patrones de diseño y combinaciones agregados pero al realizar la verificación presenta errores en el siguiente patrón de diseño.

- *Factory Method*

Y en las combinaciones

- *Strategy – Template Method*
- *Strategy – Composite*
- *Composite – Template Method*
- *Strategy – Factory Method*
- *Template Method - Factory Method*
- *Composite – Factory Method*

Esto se debe a que el algoritmo LL(1) está limitado a los patrones de diseño que fueron implementados en el desarrollo inicial de *SiVerPat* (*Observer, Iterator , Composite*) dependiendo del patrón se carga un algoritmo diferente. Es necesario restructurar la lógica de *SiVerPat* para que se implemente la forma general del algoritmo.

Además *SiVerPat* es incapaz de realizar verificación en un diagrama que fue modificado debido a que *SiVerPat* crea una instancia del diagrama y se le asigna un tipo de patrón de diseño de manera oculta al usuario. Estos tipos de patrón están inconclusas en *SiVerPat* debido a que en la codificación no se guardan en ninguna variable por lo tanto nunca cambia el tipo al cargar la regla de un patrón especifico esta falta de datos para la verificación da un resultado invalido en la verificación.

8.1 Aportaciones

De esta investigación se destacan las siguientes aportaciones:

- Modificar la herramienta SiVerPat para cargar las reglas de patrones y combinaciones de forma individual
- Agregar los botones que permiten generar los diagramas de patrones de diseño *Strategy*, *Template Method* y *Factory Method* con sus combinaciones entre estos y con el patrón *Composite*.
- Desarrollar de un intérprete que permite generar archivos XML a partir de archivos de texto plano que contengan las reglas formales de patrones de diseño o combinaciones.
- Integrar el intérprete a *SiVerPat* para cargar nuevas reglas rápidamente.

8.2 Trabajos futuros

De esta investigación se perciben los siguientes trabajos futuros:

- Reestructurar el diseño de *SiVerPat* para que el usuario pueda seleccionar que regla formal usar para la verificación.
- Proponer nuevas reglas de patrones de diseño y combinaciones de estos para agregarlos al repositorio de reglas de *SiVerPat*.
- Convertir la herramienta *SiVerPat* en servicio web que permite a diferente usuario diagramar y verificar con las reglas precargadas, reglas propuestas o reglas propuestas por otros usuarios.
- Crear un repositorio de reglas de patrones de diseño y combinaciones para que diferentes usuarios puedan usar diferentes reglas.

Bibliografia

(Blewitt, 2005) Blewitt, A. A. (2005). "Automatic verification of design patterns in Java". *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering - ASE '05*.

(Ferruci, 1996) Ferrucci, F. N. (1996). An Interpreter for Diagrammatic Languages Based on SR Grammars. *Information and Computation.*

(Fontana, 2011) Fontana, A. F. (2011). "Understanding the relevance of micro-structures for design patterns detection". *Journal of Systems and Software*.

(Gamma, 2003) Gamma, E. H. (2003). *Patrones de diseño, elemento de software orientado a objetos reutilizable.* PEARSON EDUCACIÓN, S.A.

(Gonzáles, 2010) Gonzáles, C. S. (2010). *"Estudio de factibilidad para la formulación de Reglas de Combinación de Patrones de Diseño en Arquitecturas Orientadas a Objetos".* Cuernavaca, Morelos: CENIDET.

(Liebener, 2003) Liebener, L. M. (2003). " pLinker: Relaciones con Patrones de diseño". *ISISTAN Reasearch Institute*.

(Mannhaput, 2000) Mannhaput, D. (2000). "Integration of Design Patterns into Object-Oriented Design using Rational Rose". *Universität Rostock.*

(Maplesden, 2007) Maplesden, D. J. (2007). "A Visual Language for Design Pattern Modelling and Instantiation A Visual Language for Design Pattern Modelling and Instantiation". *University of Auckland*.

(Moudam, 2012) Moudam, Z. C. (2012). "Design Pattern Support System: Help Making Decision in the Choice of Appropriate Pattern". *Procedia Technology*.

(Navarro, 2010) Navarro, I. P. (2010). " A Recommendation System to Support Design Patterns Selection". *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*.

(Nicholson, 2014) Nicholson, J. E. (2014). "Automated verification of design patterns: A case study". *Science of Computer Programming*.

(Palma, 2012) Palma, F. F.-G. (2012). "Recommendation system for design patterns in software development: An DPR overview". *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE).*

(Pérez, 2014) Pérez, Rojas. Hector Uriel. (2014). *"Reglas para la Combinación de Patrones de Diseño".* 2014: CENIDET.

(Pires, 2010) Pires, W. F. (2010). " Checking UML Design Patterns in Java Implementations". *2010 Fourth Brazilian Symposium on Software Components, Architectures and Reuse*.

(Pressman, 2010) Pressman, Robert. S. (2010). *" Ingeniería de software. Un enfoque práctico"* (Séptima edición ed.). Mc Graw Hill.

(Riehle, 2000) Riehle, D. (2000). “Framework Design. A Role Modeling Approach”. *Universität Hamburg*.

(Salazar, 2013) Salazar, Robles. Jose. (2013). *“Ambiente de Modelado de Arquitecturas de Software Conducido por Reglas de Combinación de Patrones de Diseño”.* Cuernavaca, Morelos: CENIDET.

(Sommerville, 2011) Sommerville, Ian. (2011). *Ingeniería de software*. PEARSON EDUCACIÓN, S.A.

(Xiang, 2012) Xiang, Y. Y. (2012). "The Formal Research and Application Based on Design Patterns”. *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*.

(Zavala, 2004) Zavala, R. J. (2004). “¿Por Qué Fracasan los Proyectos de Software?; Un Enfoque Organizacional”. *Congreso Nacional de Software Libre 2004*.

Anexo A. Reglas convertidas

En este anexo se muestran las reglas de los Patrones de diseño y combinaciones de las tesis (Gonzáles, 2010) y (Pérez, 2014) modificadas de la forma de que el intérprete las acepte, y las convierta a archivos XML aceptados por la herramienta SiVerPat (Salazar, 2013).

Patrones de la tesis (Gonzáles, 2010)

Patrón Observer

```
Observer = (
        {Z, S, CS, O, COS, COSX, CO},
        {subject, concreteSubject, observer, concreteObserver},
        {Inh, Agg},
        Z,
        P,
        R)
P:
        0: Z0 - <{S2, CS2, O2, COS2}, {Inh(CS2, S2), Inh(COS2, O2), Agg(S2, O2),
                Agg(COS2, CS2)}>
        1: COS0 - <{CO2, COSX2}, void >
        2: COSX0 - <{COS2}, void >
        3: COSX0 - <{void}, void >
        4: S0 - <{subject2}, void>
        5: CS0 - <{concreteSubject2}, void >
        6: O0 - <{observer2}, void >
        7: CO0 - <{concreteObserver2}, void >
R:
        0: Inh(COS0, wildcard1) - [1] {Inh(CO2, wildcard1), Inh(COSX2, wildcard1)}
                wildcard: O, observer
        1: Agg(COS0, wildcard1) - [1] {Agg(CO2, wildcard1), Agg(COSX2, wildcard1)}
                wildcard: CS, concreteSubject
        2: Inh(COSX0, wildcard1) - [2] {Inh(COS2, wildcard1)}
                wildcard: O, observer
        3: Agg(COSX0, wildcard1) - [2] {Agg(COS2, wildcard1)}
                wildcard: CS, concreteSubject
        4: Inh(COSX0, wildcard1) - [3] void
                wildcard: O, observer
        5: Agg(COSX0, wildcard1) - [3] void
                wildcard: CS, concreteSubject
        6: Inh(wildcard1, S0) - [4] {Inh(wildcard1, subject2)}
                wildcard: CS, concreteSubject
        7: Agg(S0, wildcard1) - [4] {Agg(subject2, wildcard1)}
                wildcard: O, observer
        8: Inh(CS0, wildcard1) - [5] {Inh(concreteSubject2, wildcard1)}
                wildcard: S, subject
        9: Agg(wildcard1, CS0) - [5] {Agg(wildcard1, concreteSubject2)}
                wildcard: COS, COSX, CO, concreteObserver
        10: Inh(wildcard1, O0) - [6] {Inh(wildcard1, observer2)}
                wildcard: COS, COSX, CO, concreteObserver
        11: Agg(wildcard1, O0) - [6] {Agg(wildcard1, observer2)}
                wildcard: S, subject
        12: Inh(CO0, wildcard1) - [7] {Inh(concreteObserver2, wildcard1)}
                wildcard: O, observer
        13: Agg(CO0, wildcard1) - [7] {Agg(concreteObserver2, wildcard1)}
                wildcard: CS, concreteSubject
```

Patrón Iterator

```
Iterator = (
        {Z, CN, CNX, A, CA, I, CI},
        {aggregate, concreteAggregate, iterator, concreteIterator},
        {Inh, Ins, Agg},
        Z, P, R
        )
P:
        0: Z0 - <{A2, CN2, I2}, {Inh(CN2, A2), Inh(CN2, I2)}>
        1: CN0 - <{CA2, CI2, CNX2}, {Ins(CA2, CI2), Agg(CI2, CA2)}>
        2: CNX0 - <{CN2}, void>
        3: CNX0 - <{void}, void>
        4: A0 - <{aggregate2}, void>
        5: CA0 -<{concreteAggregate2},void>
        6: I0 -<{iterator2},void>
        7: CI0 -<{concreteIterator2},void>
R:
        0: Inh(CN0, wildcard1) - [1] {Inh(CA2, wildcard1), Inh(CNX2, wildcard1)}
                wildcard: A, aggregate
        1: Inh(CN0, wildcard1) - [1] {Inh(CI2, wildcard1), Inh(CNX2, wildcard1)}
                wildcard: I, iterator
        2: Inh(CNX0, wildcard1) - [2] {Inh(CN2, wildcard1)}
                wildcard: A, I, aggregate, iterator
        3: Inh(CNX0, wildcard1) - [3] void
                wildcard: A, I, aggregate, iterator
        4: Inh(wildcard1, A0) - [4] {Inh(wildcard1, aggregate2)}
                wildcard: CN, CNX, CA, concreteAggregate
        5: Inh(CA0, wildcard1) - [5] {Inh(concreteAggregate2, wildcard1)}
                wildcard: A, aggregate
        6: Ins(CA0, wildcard1) - [5] {Ins(concreteAggregate2, wildcard1)}
                wildcard: CI, concreteIterator
        7: Agg(wildcard1, CA0) - [5] { Agg(wildcard1, concreteAggregate2)}
                wildcard: CI, concreteIterator
        8: Inh(wildcard1, I0) - [6] {Inh(wildcard1, iterator2)}
                wildcard: CN, CNX, CI, concreteIterator
        9: Inh(CI0, wildcard1) - [7] {Inh(concreteIterator2, wildcard1)}
                wildcard: I, iterator
        10: Ins(wildcard1, CI0) - [7] {Ins(wildcard1, concreteIterator2)}
                wildcard: CA, concreteAggregate
        11: Agg(CI0, wildcard1) -[7] {Agg(concreteIterator2, wildcard1)}
                wildcard: CA, concreteAggregate
```

Patrón Composite

Este patrón de diseño también es usado en la tesis de Hector Uriel P, por lo tanto solo se definirá en esta sección.

```
Composite= (
                        {Z, CE, LV, LVX, LF, CT},
                        {component, leaf, composite},
                        {Inh, Comp},
                        Z,
                        P,
                        R
)

P:
```

0: Z0 - <{CE2,LV2,CT2 },{Inh(LV2,CE2),Inh(CT2,CE2),Comp(CT2,CE2)} >
1: LV0 - <{LF2,LVX2 },void >
2: LVX0 - <{LV2 },void >
3: LVX0 - < {void},void >
4: CE0 - < {component2},void >
5: CT0 - < {composite2 },void >
6: LF0 - < {leaf2 },void >

R:

0: Inh(LV0,wildcard1) -[1]{Inh(LF2,wildcard1),Inh(LVX2,wildcard1)}
wildcard:CE,component
1: Inh(LVX0,wildcard1) -[2]{Inh(LV2,wildcard1)}
wildcard:CE,component
2: Inh(LVX0,wildcard1) -[3]void
wildcard:CE,component
3: Inh(wildcard1,CE0) -[4]{Inh(wildcard1,component2)}
wildcard:LV,LVX,LF,CT,leaf,composite
4: Comp(wildcard1,CE0) -[4]{Comp(wildcard1,component2)}
wildcard:CT,composite
5: Inh(CT0,wildcard1) -[5]{Inh(composite2,wildcard1)}
wildcard:CE,component
6: Comp(CT0,wildcard1) -[5]{Comp(composite2,wildcard1)}
wildcard:CE,component
7: Inh(LF0,wildcard1) -[6]{Inh(leaf2,wildcard1)}
wildcard:CE,component

Combinaciones de la tesis (Gonzáles, 2010)

Combiracion Observer-Composite

ObserverComposite = (
{Z, PO, PC, S, CS, O, CE, LV, LVX, CT, LF},
{subject, concreteSubject, observer, component, composite, leaf},
{Inh, Agg, Comp},
Z,
P,
R)

P:

0: Z0 - <{PO2, PC2}, {Inh(PC2, PO2), Agg(PC2, PO2)}>
1: PO0 - <{S2, CS2, O2}, {Inh(CS2, S2), Agg(S2, O2)}>
2: PC0 - <{CE2, LV2, CT2}, {Inh(LV2, CE2), Inh(CT2, CE2), Comp(CT2, CE2)}>
3: S0 - <{subject2}, void>
4: CS0 - <{concreteSubject2}, void>
5: O0 - <{observer2}, void>
6: CE0 - <{component2}, void>
7: LV0 - <{LF2, LVX2}, void>
8: LVX0 - <{LV2}, void>
9: LVX0 - <{void}, void>
10: CT0 - <{composite2}, void>
11: LF0 - <{leaf2}, void>

R:

0: Inh(wildcard1, PO0) - [1] {Inh(wildcard1, O2)}
wildcard: PC, CE, component
1: Agg(wildcard1, PO0) - [1] {Agg(wildcard1, CS2)}
wildcard: PC, LV, LF, CT, leaf, composite
2: Inh(PC0, wildcard1) - [2] {Inh(CE2, wildcard1)}
wildcard: PO, O, observer

3: Agg(PC0, wildcard1) - [2] {Agg(LV2, wildcard1), Agg(CT2, wildcard1)}
wildcard: PO, CS, concreteSubject
4: Inh(wildcard1, S0) - [3] {Inh(wildcard1, subject2)}
wildcard: CS, concreteSubject
5: Agg(S0, wildcard1) - [3] {Agg(subject2, wildcard1)}
wildcard: O, observer
6: Inh(CS0, wildcard1) - [4] {Inh(concreteSubject2, wildcard1)}
wildcard: S, subject
7: Agg(wildcard1, CS0) - [4] {Agg(wildcard1, concreteSubject2)}
wildcard: PC, LV, LVX, LF, leaf
8: Inh(wildcard1, O0) - [5] {Inh(wildcard1, observer2)}
wildcard: PC, CE, component
9: Agg(wildcard1, O0) - [5] {Agg(wildcard1, observer2)}
wildcard: S, subject
10: Inh(CE0, wildcard1) - [6] {Inh(component0, wildcard1)}
wildcard: PO, O, observer
11: Inh(wildcard1, CE0) - [6] {Inh(wildcard1, component2)}
wildcard: LV, LVX, LF, CT, leaf, composite
12: Comp(wildcard1, CE0) - [6] {Comp(wildcard1, component2)}
wildcard: CT, composite
13: Inh(LV0, wildcard1) - [7] {Inh(LF2, wildcard1), Inh(LVX2, wildcard1)}
wildcard: CE, component
14: Agg(LV0, wildcard1) - [7] {Agg(LF2, wildcard1), Agg(LVX2, wildcard1)}
wildcard: PO, CS, concreteSubject
15: Inh(LVX0, wildcard1) - [8] {Inh(LV2, wildcard1)}
wildcard: CE, component
16: Agg(LVX0, wildcard1) - [8] {Agg(LV2, wildcard1)}
wildcard: PO, CS, concreteSubject
17: Inh(LVX0, wildcard1) - [9] void
wildcard: CE, component
18: Agg(LVX0, wildcard1) - [9] void
wildcard: PO, CS, concreteSubject
19: Inh(CT0, wildcard1) - [10]{ Inh(composite2, wildcard1)}
wildcard: CE, component
20: Comp(CT0, wildcard1) - [10]{ Comp(composite2, wildcard1)}
wildcard: CE, component
21: Agg(CT0, wildcard1) - [10]{ Agg(composite2, wildcard1)}
wildcard: PO, CS, concreteSubject
22: Inh(LF0, wildcard1) - [11]{ Inh(leaf2, wildcard1)}
wildcard: CE, component
23: Agg(LF0, wildcard1) - [11]{ Agg(leaf2, wildcard1)}
wildcard: PO, CS, concreteSubject

Combinación con Iterator

La herramienta SiVerPat usa las reglas del patrón Iterator para las combinaciones con Iterator.

Patrónes de la tesis (Pérez, 2014)

Patrón Strategy estatico

```
Strategy= (
                    {Z, C, S, CS},
                    {context, Strategy, concreteStrategy},
                    {Inh, Comp},
                    Z,
                    P,
```

R
)

P:
0: Z0 - < {C2,S2,CS2 },{Inh(CS2,S2),Comp(C2,S2)} >
1: C0 - < {context2},void >
2: S0 - < {Strategy2},void >
3: CS0 - < {concreteStrategy2},void >

R:
1: Comp(C0,wildcard1) -[1]{Comp(context2,wildcard1)}
wildcard:S,Strategy
2: Comp(wildcard1,S0) -[2]{Comp(wildcard1,Strategy2)}
wildcard:C,context
3: Inh(wildcard1,S0) -[2]{Inh(wildcard1,Strategy2)}
wildcard: CS,concreteStrategy
4: Inh(CS0,wildcard1) -[3]{Inh(concreteStrategy2,wildcard1)}
wildcard:S,Strategy

Patrón Strategy

Strategy= (
{Z, C, S, CS, CSS, CSSX},
{context, Strategy, concreteStrategy},
{Inh, Comp},
Z,
P,
R
)

P:
0: Z0 - < {C2,S2,CSS2 },{Inh(CSS2,S2),Comp(C2,S2)} >
1: CSS0 - < {CS2,CSSX2 },void >
2: CSSX0 - < {CSS2},void >
3: CSSX0 - < {void},void >
4: CS0 - < {concreteStrategy2},void >
5: S0 - < {Strategy2},void >
6: C0 - < {context2},void >

R:

0: Inh(CSS0,wildcard1) -[1]{Inh(CS2,wildcard1),Inh(CSSX2,wildcard1)}
wildcard:S,Strategy
1: Inh(CSSX0,wildcard1) -[2]{Inh(CSS2,wildcard1)}
wildcard:S,Strategy
2: Inh(CSSX0,wildcard1) -[3]void
wildcard:S,Strategy
3: Inh(CS0,wildcard1) -[4]{Inh(concreteStrategy2,wildcard1)}
wildcard:S,Strategy
4: Comp(wildcard1,S0) -[5]{Comp(wildcard1,Strategy2)}
wildcard:C,context
5: Comp(C0,wildcard1) -[6]{Comp(context2,wildcard1)}
wildcard:S,Strategy

Patrón Template Method

```
TemplateMethod= (
                        {Z, AC, CCD, CCX, CC},
                        {abstractClass, concreteClass},
                        {Inh},
                        Z,
                        P,
                        R
)

P:
        0: Z0 - < {AC2,CCD2 },{Inh(CCD2,AC2 )}  >
        1: CCD0 - < {CC2,CCX2 }, void>
        2: CCX0 - < {CCD2 },void >
        3: CCX0 - < {void},void >
                4: AC0 - < {abstractClass2},void >
5: CC0 - < {concreteClass2},void >



R:
        0: Inh(CCD0,wildcard1 )  -[1]{Inh(CC2,wildcard1 ),Inh(CCX2,wildcard1 )}
                wildcard:AC,abstractClass
        1: Inh(CCX0,wildcard1 )  -[2]{Inh(CCD2,wildcard1)}
                wildcard:AC,abstractClass
        2: Inh(CCX0,wildcard1 )  -[3]  void
                wildcard:AC,abstractClass
        3: Inh(wildcard1,AC0 )  -[4]{Inh(wildcard1,abstractClass2)}
                wildcard:CC,concreteClass
        4: Inh(CC0,wildcard1 )  -[5]{Inh(concreteClass2,wildcard1)}
                wildcard:AC,abstractClass
```

Patrón Template Method estatico

```
TemplateMethod= (
                        {Z, AC, CC},
                        {abstractClass, concreteClass},
                        {Inh},
                        Z,
                        P,
                        R
)

P:
        0: Z0 - < {AC2,CC2 },{Inh(CC2,AC2 )}  >
        1: AC0 - < {abstractClass2},void >
        2: CC0 - < {concreteClass2},void >
R:

        1: Inh(wildcard1,AC0 )  -[1]{Inh(wildcard1,abstractClass2)}
                wildcard:CC,concreteClass
        2: Inh(CC0,wildcard1 )  -[2]{Inh(concreteClass2,wildcard1)}
                wildcard:AC,abstractClass
```

Patrón Factory Method

```
FactoryMethod = (
                        {Z, CPS, CPSX, P, C, CP, CC},
                        {product, creator, concreteProduct, concreteCreator},
                        {Inh, Ins},
                        Z,
                        P,
                        R
)

P:
        0: Z0 - <{P2,C2,CPS2 },{Inh(CPS2,P2 ),Inh(CPS2,C2 )}  >
        1: CPS0 - <{CP2,CPSX2,CC2 },{Ins(CC2,CP2 )}  >
        2: CPSX0 - <{CPS2 },void  >
        3: CPSX0 - < {void},void >
        4: P0 - < {product2 },void >
        5: C0 - < {creator2},void >
        6: CP0 - < {concreteProduct2 },void >
        7: CC0 - < {concreteCreator2},void >


R:
        0: Inh(CPS0,wildcard1 )  -[1]{Inh(CP2,wildcard1)}
                wildcard:P,product
        1: Inh(CPS0,wildcard1 )  -[1]{Inh(CPSX2,wildcard1)}
                wildcard:P,product
        2: Inh(CPS0,wildcard1 )  -[1]{Inh(CC2,wildcard1)}
                wildcard:C,creator
        3: Inh(CPSX0,wildcard1 )  -[2]{Inh(CPS2,wildcard1)}
                wildcard:P,product
        4: Inh(CPSX0,wildcard1 )  -[3]void
                wildcard:P,product
        5: Inh(wildcard1,P0 )  -[4]{Inh(wildcard1,product2)}
                wildcard:CPS,CP,concreteProduct
        6: Inh(wildcard1,C0 )  -[5]{Inh(wildcard1,creator2)}
                        wildcard:CPS,CC,concreteCreator
        7: Inh(CP0,wildcard1 )  -[6]{Inh(concreteProduct2,wildcard1)}
                wildcard:P,product
        8: Ins(wildcard1,CP0 )  -[6]{Ins(wildcard1,concreteProduct2)}
                wildcard:CC,concreteCreator
        9: Inh(CC0,wildcard1 )  -[7]{Inh(concreteCreator2,wildcard1)}
                wildcard:C,creator
        10: Ins(CC0,wildcard1 )  -[7]{Ins(concreteCreator2,wildcard1)}
                wildcard:CP,concreteProduct
```

Patrón Factory Method estático

```
FactoryMethod = (
                        {Z, C, CC, P, CP},
                        {creator,concreteCreator, product,  concreteProduct },
                        {Inh, Ins},
                        Z,
                        P,
                        R
)
```

P:

0: Z0 - <{C2,CC2,P2,CP2 },{Inh(CC2,C2),Inh(CP2,P2),Ins(CC2,CP2)} >
1: C0 - < {creator2},void >
2: CC0 - < {concreteCreator2},void >
3: P0 - < {product2 },void >
4: CP0 - < {concreteProduct2 },void >

R:

1: Inh(wildcard1,C0) -[1]{Inh(wildcard1,creator2)}
wildcard:CC,concreteCreator
2: Inh(CC0,wildcard1) -[2]{Inh(concreteCreator2,wildcard1)}
wildcard:C,creator
3: Ins(CC0,wildcard1) -[2]{Ins(concreteCreator2,wildcard1)}
wildcard:CP,concreteProduct
4: Inh(wildcard1,P0) -[3]{Inh(wildcard1,product2)}
wildcard:CP,concreteProduct
5: Inh(CP0,wildcard1) -[4]{Inh(concreteProduct2,wildcard1)}
wildcard:P,product
6: Ins(wildcard1,CP0) -[4]{Ins(wildcard1,concreteProduct2)}
wildcard:CC,concreteCreator

Combinaciones de la tesis (Pérez, 2014)

Combinación Strategy- Composite

```
StrategyComposite = (
                        {Z, PS, PC, C, S, CSS, CSSX, CS, CE, LV, LVX, CT, LF},
                        {context, strategy, concreteStrategy, component, composite, leaf},
                        {Inh,Comp},
                        Z,
                        P,
                        R
)
```

P:

0: Z0 - <{PS2,PC2 },{Inh(PC2,PS2)} >
1: PS0 - < {C2,S2,CSS2 },{Inh(CSS2,S2),Comp(C2,S2)} >
2: PC0 - <{CE2,LV2,CT2 },{Inh(LV2,CE2),Inh(CT2,CE2),Comp(CT2,CE2)} >
3: C0 - < {context2},void >
4: S0 - < {strategy2},void >
5: CSS0 - < {CS2,CSSX2 },void >
6: CSSX0 - < {CSS2},void >
7: CSSX0 - < {void},void >
8: CS0 - < {concreteStrategy2},void >
9: CE0 - < {component2},void >
10: LV0 - <{LF2,LVX2 },void >
11: LVX0 - <{LV2 },void >
12: LVX0 - < {void},void >
13: CT0 - < {composite2 },void >
14: LF0 - < {leaf2 },void >

R:

0: Inh(wildcard1,PS0) -[1]{Inh(wildcard1,S2)}
wildcard:PC,CE,component

1: Inh(PC0,wildcard1) -[2]{Inh(CE2,wildcard1)}
wildcard:PC,S,strategy
2: Comp(C0,wildcard1) -[3]{Inh(context2,wildcard1)}
wildcard:S,strategy
3: Inh(wildcard1,S0) -[4]{Inh(wildcard1,strategy2)}
wildcard:CSS,CS,concreteStrategy,PC,CE,component
4: Comp(wildcard1,S0) -[4]{Comp(wildcard1,strategy2)}
wildcard:C,context
5: Inh(CSS0,wildcard1) -[5]{Inh(CS2,wildcard1),Inh(CSSX2,wildcard1)}
wildcard:S,strategy
6: Inh(CSSX0,wildcard1) -[6]{Inh(CSS2,wildcard1)}
wildcard:S,strategy
7: Inh(CSSX0,wildcard1) -[7]void
wildcard:S,strategy
8: Inh(CS0,wildcard1) -[8]{Inh(concreteStrategy2,wildcard1)}
wildcard:S,strategy
9: Inh(CE0,wildcard1) -[9]{Inh(component2,wildcard1)}
wildcard:PS,S,strategy
10: Inh(wildcard1,CE0) -[9]{Inh(wildcard1,component2)}
wildcard:LV,LVX,LF,CT,leaf,composite
11: Comp(wildcard1,CE0) -[9]{Comp(wildcard1,component2)}
wildcard:CT,composite
12: Inh(LV0,wildcard1) -[10]{Inh(LF2,wildcard1),Inh(LVX2,wildcard1)}
wildcard:CE,component
14: Inh(LVX0,wildcard1) -[11]{Inh(LV2,wildcard1)}
wildcard:CE,component
16: Inh(LVX0,wildcard1) -[12]void
wildcard:S,strategy
17: Inh(CT0,wildcard1) -[13]{Inh(composite2,wildcard1)}
wildcard:CE,component
18: Comp(CT0,wildcard1) -[13]{Comp(composite2,wildcard1)}
wildcard:CE,component
19: Inh(LF0,wildcard1) -[14]{Inh(leaf2,wildcard1)}
wildcard:CE,component

Combinación Strategy – Template Method

StrategyTemplate = (
{Z, PSPTM, C, S-AC, CSD-CCD, CS-CC, CSDX-CCDX},
{context,strategyAbstractClass, concreteStrategyconcreteClass},
{Inh,Comp},
Z,
P,
R
)

P:
0: Z0 - < {PSPTM2 }, void >
1: PSPTM0 - < {C2,S-AC2,CSD-CCD2 },{Inh(CSD-CCD2,S-AC2),Comp(C2,S-AC2)} >
2: CSD-CCD0 - < {CS-CC2,CSDX-CCDX2 },void >
3: CSDX-CCDX0 - < {CSD-CCD2 },void >
4: CSDX-CCDX0 - < {void},void >
5: C0 - < {context2 },void >
6: S-AC0 - < {strategyAbstractClass2},void >
7: CS-CC0 - < {concreteStrategyconcreteClass2},void >

R:

0: Inh(CSD-CCD0,wildcard1) -[2]{Inh(CS-CC2,wildcard1),Inh(CSDX-CCDX2,wildcard1)}
wildcard:S-AC,strategyAbstractClass
1: Inh(CSDX-CCDX0,wildcard1) -[3]{Inh(CSD-CCD2,wildcard1)}
wildcard:S-AC,strategyAbstractClass
2: Inh(CSDX-CCDX0,wildcard1) -[4] void
wildcard:S-AC,strategyAbstractClass
3: Comp(C0,wildcard1) -[5]{Comp(context2,wildcard1)}
wildcard:S-AC,strategyAbstractClass
4: Inh(wildcard1,S-AC0) -[6]{Inh(wildcard1,strategyAbstractClass2)}
wildcard:CSD-CCD,concreteStrategyconcreteClass
5: Comp(wildcard1,S-AC0) -[6]{Comp(wildcard1,strategyAbstractClass2)}
wildcard:C,context
6: Inh(CS-CC0,wildcard1) -[7]{Inh(concreteStrategyconcreteClass2,wildcard1)}
wildcard:S-AC,strategyAbstractClass

Combinación Strategy – Factory Method

```
StrategyFactory = (
                        {Z, PS, PFM, CPS, CPSX, P, C, CP, CC, CTX},
                        {strategyProduct, concreteProduct, creator, concreteCreator, context },
                        {Inh, Ins, Comp},
                        Z,
                        P,
                        R
)

P:
        0: Z0 - < {PS2,PFM2 },{ Comp(PS2,PFM2 )}  >
        1: PS0 - < {CTX2},void >
        2: PFM0 - <{P2,C2,CPS2 },{Inh(CPS2,P2 ),Inh(CPS2,C2 )}  >
        3: CPS0 - <{CP2,CPSX2,CC2 },{Ins(CC2,CP2 )}  >
        4: CPSX0 - < {CPS2},void >
        5: CPSX0 - < {void},void >
        6: P0 - < {product2 },void >
        7: C0 - < {creator2},void >
        8: CP0 - < {concreteProduct2 },void >
        9: CC0 - < {concreteCreator2},void >
        10: CTX0 - < {context2},void >

R:
0: Comp(PS0,wildcard1 ) -[1]{Comp(CTX2,wildcard1 )}
        wildcard:PFM,P,product
1: Comp(wildcard1,ST0 ) -[2]{Comp(wildcard1,P2)}
        wildcard:CTX,context
2: Inh(CPS0,wildcard1 ) -[3]{Inh(CP2,wildcard1)}
        wildcard:P,product
3: Inh(CPS0,wildcard1 ) -[3]{Inh(CPSX2,wildcard1)}
        wildcard:P,product
4: Inh(CPS0,wildcard1 ) -[3]{Inh(CC2,wildcard1)}
        wildcard:C,creator
5: Inh(CPSX0,wildcard1 ) -[4]{Inh(CPS2,wildcard1)}
        wildcard:P,product
6: Inh(CPSX0,wildcard1 ) -[5]void
        wildcard:P,product
7: Inh(wildcard1,P0 ) -[6]{Inh(wildcard1,product2)}
        wildcard:CPS,CP,concreteProduct
8: Inh(wildcard1,C0 ) -[7]{Inh(wildcard1,creator2)}
```

wildcard:CPS,CC,concreteCreator
9: Inh(CP0,wildcard1) -[8]{Inh(concreteProduct2,wildcard1)}
wildcard:P,product
10: Ins(wildcard1,CP0) -[8]{Ins(wildcard1,concreteProduct2)}
wildcard:CC,concreteCreator
11: Inh(CC0,wildcard1) -[9]{Inh(concreteCreator2,wildcard1)}
wildcard:C,creator
12: Ins(CC0,wildcard1) -[9]{Ins(concreteCreator2,wildcard1)}
wildcard:CP,concreteProduct
13: Comp(CTX0,wildcard1) -[10]{Comp(context2,wildcard1)}
wildcard:PFM,P,product

Combinación Composite- Template Method

```
CompositeTemplate = (
                        {Z, LFD-CCD, LFDX-CCDX, CE-AC, CT-CC, LF-CC },
                        {componentAbstractClass, leafConcreteClass, compositeConcreteClass},
                        {Inh, Comp},
                        Z,
                        P,
                        R
)

P:
        0: Z0 - < {CE-AC2,LFD-CCD2,CT-CC2 },{Inh(LFD-CCD2,CE-AC2 ),Inh(CT-CC2,CE-
        AC2 ),Comp(CT-CC2,CE-AC2)} >
        1: LFD-CCD0 - < {LF-CC2,LFDX-CCDX2 }, void>
        2: LFDX-CCDX0 - < {LFD-CCD2 },void >
        3: LFDX-CCDX0 - < {void},void >
        4: CE-AC0 - < {componentAbstractClass2},void >
        5: CT-CC0 - < {compositeConcreteClass2},void >
                6: LF-CC0 - < {leafConcreteClass2},void >

R:
0: Inh(LFD-CCD0,wildcard1 ) -[1]{Inh(LF-CC2,wildcard1 ),Inh(LFDX-CCDX2,wildcard1)}
        wildcard:CE-AC,componentAbstractClass
1: Inh(LFDX-CCDX0,wildcard1 ) -[2]{Inh(LFD-CCD2,wildcard1)}
        wildcard:CE-AC,componentAbstractClass
2: Inh(LFDX-CCDX0,wildcard1 ) -[3]void
        wildcard:CE-AC,componentAbstractClass
3: Inh(wildcard1,CE-AC0 ) -[4]{Inh(wildcard1,componentAbstractClass2)}
        wildcard:LFD-CCD,LF-CC,CT-CC,leafConcreteClass,compositeConcreteClass
4: Comp(wildcard1,CE-AC0 ) -[4]{Comp(wildcard1,componentAbstractClass2)}
        wildcard:CT-CC,compositeConcreteClass
5: Inh(CT-CC0,wildcard1 ) -[5]{Inh(compositeConcreteClass2,wildcard1)}
        wildcard:CE-AC,componentAbstractClass
6: Comp(CT-CC0,wildcard1 ) -[5]{Comp(compositeConcreteClass2,wildcard1)}
        wildcard:CE-AC,componentAbstractClass
7: Inh(LF-CC0,wildcard1 ) -[6]{Inh(leafConcreteClass2,wildcard1)}
        wildcard:CE-AC,componentAbstractClass
```

Combinación Composite – Factory Method

CompositeFactory = (
{Z, CPLS, CPLSX, CPCS, CPCSX, P, CP, CPL, CCP, CC, CPC, CCC},
{product, creatorProduct, concreteProductLeaf, concreteCreatorProduct, creatorComposite , concreteProductComposite, concreteCreatorComposite},
{Inh, Ins},
Z,
P,
R
)

P:
0: Z0 -<{P2,CP2,CPLS2,CC2,CPCS2 },{Inh(CPLS2,P2),Inh(CPLS2,CP2),Inh(CPCS2,P2),Inh(CPCS2,CC2)} >
1: CPLS0 - <{CPL2,CPLSX2,CCP2 },{Ins(CCP2,CPL2)} >
2: CPLSX0 - <{CPLS2 },void >
3: CPLSX0 - < {void},void >
4: CPCS0 - <{CPC2,CPCSX2,CCC2 },{Ins(CCC2,CPC2)} >
5: CPCSX0 - <{CPCS2 },void >
6: CPCSX0 - < {void},void >
7: P0 - < {product2},void >
8: CP0 - < {creatorProduct2},void >
9: CPL0 - < {concreteProductLeaf2 },void >
10: CCP0 - < {concreteCreatorProduct2},void >
11: CC0 - < {creatorComposite2},void >
12: CPC0 - < {concreteProductComposite2 },void >
13: CCC0 - < {concreteCreatorComposite2},void >

R:
0: Inh(CPLS0,wildcard1) -[1]{Inh(CPL2,wildcard1)}
wildcard:P,product
1: Inh(CPLS0,wildcard1) -[1]{Inh(CPLSX2,wildcard1)}
wildcard:P,product
2: Inh(CPLS0,wildcard1) -[1]{Inh(CCP2,wildcard1)}
wildcard:CP,creatorProduct
3: Inh(CPLSX0,wildcard1) -[2]{Inh(CPLS2,wildcard1)}
wildcard:P,product
4: Inh(CPLSX0,wildcard1) -[3]void
wildcard:P,product
5: Inh(CPCS0,wildcard1) -[4]{Inh(CPC2,wildcard1)}
wildcard:P,product
6: Inh(CPCS0,wildcard1) -[4]{Inh(CPCSX2,wildcard1)}
wildcard:P,product
7: Inh(CPCS0,wildcard1) -[4]{Inh(CCC2,wildcard1)}
wildcard:CC,creatorComposite
8: Inh(CPCSX0,wildcard1) -[5]{Inh(CPCS2,wildcard1),Inh(CPCS2,wildcard1)}
wildcard:P,product
9: Inh(CPCSX0,wildcard1) -[6]void
wildcard:P,product
10: Inh(wildcard1,P0) -[7]{Inh(wildcard1,product2)}
wildcard:CPLS,CPL,CPLSX,CPCS,CCP,CCPCSX,CPC,CCC,concreteCreatorComposite,concreteProductLeaf
11: Inh(wildcard1,CP0) -[8]{Inh(wildcard1,creatorProduct2)}

wildcard:CPLS,CCP,concreteCreatorProduct
12: Inh(CPL0,wildcard1) -[9]{Inh(concreteProductLeaf2,wildcard1)}
wildcard:P,product
13: Ins(wildcard1,CPL0) -[9]{Ins(wildcard1,concreteProductLeaf2)}
wildcard:CCP,concreteCreatorProduct
14: Inh(CCP0,wildcard1) -[10]{Inh(concreteCreatorProduct2,wildcard1)}
wildcard:CP,creatorProduct
15: Ins(CCP0,wildcard1) -[10]{Ins(concreteCreatorProduct2,wildcard1)}
wildcard:CPL,concreteProductLeaf
16: Inh(wildcard1,CC0) -[11]{Inh(wildcard1,creatorComposite2)}
wildcard:CPCS,CCC,concreteCreatorComposite
17: Inh(CPC0,wildcard1) -[12]{Inh(concreteProductComposite2,wildcard1)}
wildcard:P,product
18: Ins(wildcard1,CPC0) -[12]{Ins(wildcard1,concreteProductComposite2)}
wildcard:CCC,concreteCreatorComposite
19: Inh(CCC0,wildcard1) -[13]{Inh(concreteCreatorComposite2,wildcard1)}
wildcard:CC,creatorComposite
20: Ins(CCC0,wildcard1) -[13]{Ins(concreteCreatorComposite2,wildcard1)}
wildcard:CPC,concreteProductComposite

Combinación Template Method – Factory Method

```
TemplateFactory = (
                    {Z, PTM-PFM, CPS, CPSX, P, C, CP,CC},
                    {product, concreteProduct,creator,concreteCreator },
                    {Inh,Ins},
                    Z,
                    P,
                    R
)

P:
        0: Z0 - < {PTM-PFM2},void >
        1: PTM-PFM0 - < {P2,C2,CPS2 },{Inh(CPS2,P2),Inh(CPS2,C2)}>
        2: CPS0 - <{CP2,CPSX2,CC2 },{Ins(CC2,CP2 )}  >
        3: CPSX0 - <{CPS2 }, void  >
        4: CPSX0 - < {void},void >
        5: P0 - < {product2},void >
        6: C0 - < {creator2},void >
        7: CP0 - < {concreteProduct2},void >
        8: CC0 - < {concreteCreator2},void >



R:
0: Inh(CPS0,wildcard1 ) -[2]{Inh(CP2,wildcard1)}
                    wildcard:P,product
1: Inh(CPS0,wildcard1 ) -[2]{Inh(CPSX2,wildcard1)}
                    wildcard:P,product
2: Inh(CPS0,wildcard1 ) -[2]{Inh(CC2,wildcard1)}
                    wildcard:C,creator
3: Inh(CPSX0,wildcard1 ) -[3]{Inh(CPS2,wildcard1)}
                    wildcard:P,product
4: Inh(CPSX0,wildcard1 ) -[4]void
                    wildcard:P,product
5: Inh(wildcard1,P0 ) -[5]{Inh(wildcard1,product2)}
                    wildcard:CPS,CP,concreteProduct
```

6: Inh(wildcard1,C0) -[6]{Inh(wildcard1,creator2)}
wildcard:CPS,CC,concreteCreator
7: Inh(CP0,wildcard1) -[7]{Inh(concreteProduct2,wildcard1)}
wildcard:P,product
8: Ins(wildcard1,CP0) -[7]{Ins(wildcard1,concreteProduct2)}
wildcard:CC,concreteCreator
9: Inh(CC0,wildcard1) -[8]{Inh(concreteCreator2,wildcard1)}
wildcard:C,creator
10: Ins(CC0,wildcard1) -[8]{Ins(concreteCreator2,wildcard1)}
wildcard:CP,concreteProduct

Anexo B. Resultado de las pruebas del Intérprete

En este anexo se describen los resultados obtenidos en las pruebas de aceptación del intérprete

Ejecución de prueba PAI_01

Nombre prueba:	**PAI_01**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_01		
Objetivo:	Comprobar que el usuario puede cargar un archivo TXT		
Condiciones de Ambiente:	Iniciar el programa		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Cargar archivo “strategy.txt”	Se cargó el contenido del archivo en el intérprete.	
Observaciones			

Tabla 29. Ejecución de prueba PAI_01

Ejecución de prueba PAI_02

Nombre prueba:	**PAI_02**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_03		
Objetivo:	Comprobar que el usuario puede guardar el archivo XML		
Condiciones de Ambiente:	Se ha concluido la traducción del archivo de forma correcta		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Guardar archivo “strategy.xml”	Se guardó el archivo de manera correcta.	
Observaciones			

Tabla 30. Ejecución de prueba PAI_02

Ejecución de prueba PAI_03

Nombre prueba:	**PAI_03**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_02		
Objetivo:	Comprobar que el intérprete traduzca de forma correcta las reglas de los patrones que aparecen en la tesis [Gonz,2010]		
Condiciones de Ambiente:	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Contenido del archivo “observer.txt”	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones			
Instancia 2	Contenido del archivo “iterator.txt”	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones			
Instancia 3	Contenido del archivo “composite.txt”	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones			

Tabla 31. Ejecución de prueba PAI_03

Ejecución de prueba PAI_04

Nombre prueba:	**PAI_04**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_02		
Objetivo:	Comprobar que el intérprete traduzca de forma correcta las reglas de las combinaciones de patrones que aparecen en la tesis [Gonz,2010]		
Condiciones de Ambiente:	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Instancias de la prueba			

No. Instancia	Entrada	Resultado
Instancia 1	Contenido del archivo “ObserverComposite.txt”	Se tradujo el contenido del archivo a XML correctamente.
Observaciones		
Instancia 2	Contenido del archivo “iterator.txt”	Se tradujo el contenido del archivo a XML correctamente.
Observaciones	SiVerPat usa la regla de Iterator para las combinaciones de patrones que usan Iterator.	

Tabla 32. Ejecución de prueba PAI_04

Ejecución de prueba PAI_05

Nombre prueba:	**PAI_05**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_02		
Objetivo:	Comprobar que el intérprete traduzca de forma correcta las reglas de los patrones que aparecen en la tesis [Pérez, 2014] en su forma dinámica.		
Condiciones de Ambiente:	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Contenido del archivo “strategy.txt” en su forma dinámica	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones			
Instancia 2	Contenido del archivo “templateMethod.txt” en su forma dinámica	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.		
Instancia 3	Contenido del archivo “factoryMethod.txt” en su forma dinámica	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.		

Tabla 33. Ejecución de prueba PAI_05

Ejecución de prueba PAI_06

Nombre prueba:	**PAI_06**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_02		
Objetivo:	Comprobar que el intérprete traduzca de forma correcta las reglas de los patrones que aparecen en la tesis [Pérez, 2014] en su forma estática.		
Condiciones de Ambiente:	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Contenido del archivo “strategy.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones			
Instancia 2	Contenido del archivo “templateMethod.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.		
Instancia 3	Contenido del archivo “factoryMethod.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.	
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.		

Tabla 34. Ejecución de prueba PAI_06

Ejecución de prueba PAI_07

Nombre prueba:	**PAI_07**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_02		
Objetivo:	Comprobar que el intérprete traduzca de forma correcta las reglas de las combinaciones de patrones de diseño que aparecen en la tesis [Pérez, 2014].		
Condiciones de Ambiente:	Se ha cargado el archivo que contiene las reglas formales al interprete correctamente		
Instancias de la prueba			

No. Instancia	Entrada	Resultado
Instancia 1	Contenido del archivo “StrategyComposite.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.
Observaciones		
Instancia 2	Contenido del archivo “StrategyTemplate.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.	
Instancia 3	Contenido del archivo “StrategyFactory.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.	
Instancia 4	Contenido del archivo “CompositeFactory.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.	
Instancia 5	Contenido del archivo “CompositeTemplate.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.	
Instancia 6	Contenido del archivo “TemplateFactory.txt” en su forma estática	Se tradujo el contenido del archivo a XML correctamente.
Observaciones	Se tiene que eliminar la sección que describe el comportamiento.	

Anexo C. Resultado del plan de pruebas

En este anexo se describen los resultados obtenidos del plan de pruebas

Ejecución de prueba PA_01

Nombre prueba:	**PA_01**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_01		
Objetivo:	Comprobar que el usuario puede crear un nuevo proyecto		
Condiciones de Ambiente:	Se ha iniciado la aplicación.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Usuario realiza el proceso de un nuevo proyecto	Se muestra una nueva ventana	
Observaciones			

Tabla 35. Ejecución de prueba PA_01

Ejecución de prueba PA_02

Nombre prueba:	**PA_02**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_02		
Objetivo:	Comprobar que el usuario puede guardar un proyecto nuevo.		
Condiciones de Ambiente:	Se ha iniciado la aplicación.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	• Nombre proyecto • Ubicación del archivo	Se guardó el archivo de manera correcta.	
Observaciones			
Instancia 2	• Nombre proyecto • Ubicación del archivo	El archivo se guardaba pero con errores	

Observaciones		
Instancia 3	• Nombre proyecto • Ubicación del archivo	No muestra la ventana de aviso
Observaciones		
Instancia 4	• Nombre proyecto • Ubicación del archivo	No muestra la ventana para cambiar el nombre
Observaciones		
Instancia 5	• Nombre proyecto • Ubicación del archivo	No se creó ningún archivo.
Observaciones		

Tabla 36. Ejecución de prueba PA_02

Ejecución de prueba PA_03

Nombre prueba:	**PA_03**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_03		
Objetivo:	Comprobar que el usuario puede abrir un proyecto nuevo.		
Condiciones de Ambiente:	Los casos de prueba PA_01 y PA_02 se ejecutaron con éxito		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Archivo “proyecto01.xml”	Se abre el archivo	
Observaciones	En ocasiones, el diagrama aparece con objetos no visibles, se tiene que refrescar la ventana para corregir este problema.		
Instancia 2	Archivo “proyecto01.xml”	No se abre el archivo	
Observaciones			
Instancia 3	Archivo “proyecto01ERROR.xml”	No se muestra la ventana de archivo invalido	
Observaciones	El sistema entra en una excepción y se bloquea		

Tabla 37. Ejecución de prueba PA_03

Ejecución de prueba PA_04

Nombre prueba:	**PA_04**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_04		
Objetivo:	Comprobar que el usuario puede abrir un proyecto nuevo.		
Condiciones de Ambiente:	Los casos de prueba PA_01 y PA_02 se ejecutaron con éxito		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Archivo "proyecto01.xml"	Se guardó correctamente un proyecto con el nombre que tenía asignado.	
Observaciones			

Tabla 38. Ejecución de prueba PA_04

Ejecución de prueba PA_05

Nombre prueba:	**PA_05**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_05		
Objetivo:	Comprobar que el usuario es capaz de agregar elementos a un diagrama de clases de UML, dentro de un proyecto creado previamente.		
Condiciones de Ambiente:	El caso de prueba PA_01 se ejecutó con éxito.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	El usuario selecciona y dibuja el elemento "Clase"	Se dibujó el elemento de forma correcta	
Observaciones			
Instancia 2	El usuario selecciona y dibuja la relación "Herencia"	Se dibujó la relación de forma correcta.	

Observaciones		
Instancia 3	El usuario selecciona y dibuja la relación “Agregación”	Se dibujó la relación de forma correcta.
Observaciones		
Instancia 4	El usuario selecciona y dibuja la relación “Composición”	Se dibujó la relación de forma correcta.
Observaciones		
Instancia 5	El usuario selecciona y dibuja la relación “Relación Vacia”	Se dibujó la relación de forma correcta.
Observaciones		
Instancia 6	El usuario selecciona y dibuja el elemento “Nota”	Se dibujó el elemento de forma correcta
Observaciones		
Instancia 7	El usuario selecciona, pero no marca un destino para una relación	No se dibujó la relación
Observaciones		

Tabla 39. Ejecución de prueba PA_05

Ejecución de prueba PA_06

Nombre prueba:	**PA_06**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_06		
Objetivo:	Comprobar que el usuario es capaz de modificar elementos de un diagrama de clases de UML, dentro de un proyecto creado previamente.		
Condiciones de Ambiente:	Los casos de prueba PA_01 y PA_05 se ejecutaron con éxito.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	El usuario selecciona y modifica el elemento “Clase”	No cambio el nombre, ni el ámbito.	
Observaciones			
Instancia 2	El usuario selecciona y módica la relación “Herencia”	No cambio el tipo de relación	

Observaciones		
Instancia 3	El usuario selecciona y modifica la relación “Agregación”	No cambio el tipo de relación
Observaciones		
Instancia 4	El usuario selecciona y modifica la relación “Composición”	No cambio el tipo de relación
Observaciones		
Instancia 5	El usuario selecciona y modifica la relación “Relación Vacia”	No cambio el tipo de relación
Observaciones		
Instancia 6	El usuario selecciona y modifica el elemento “Nota”	No cambio el tipo de relación
Observaciones		
Instancia 7	El usuario selecciona, pero no marca un destino para una relación	No hubo cambio
Observaciones		

Tabla 40. Ejecución de prueba PA_06

Ejecución de prueba PA_07

Nombre prueba:	**PA_07**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_07		
Objetivo:	Comprobar que el usuario es capaz de eliminar elementos de un diagrama de clases de UML, dentro de un proyecto creado previamente.		
Condiciones de Ambiente:	Los casos de prueba PA_01 y PA_05 se ejecutaron con éxito.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	El usuario selecciona y elimina el elemento “Clase”	Se eliminó el elemento.	
Observaciones			
Instancia 2	El usuario selecciona y elimina la relación “Herencia”	Se eliminó la relación.	

Observaciones		
Instancia 3	El usuario selecciona y elimina la relación “Agregación”	Se eliminó la relación.
Observaciones		
Instancia 4	El usuario selecciona y elimina la relación “Composición”	Se eliminó la relación.
Observaciones		
Instancia 5	El usuario selecciona y elimina la relación “Relación Vacia”	Se eliminó la relación.
Observaciones		
Instancia 6	El usuario selecciona y elimina el elemento “Nota”	Se eliminó el elemento.
Observaciones		

Tabla 41. Ejecución de prueba PA_07

Ejecución de prueba PA_08

Nombre prueba:	**PA_08**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_08		
Objetivo:	Comprobar que la herramienta es capaz de verificar patrones de diseño en el diagrama.		
Condiciones de Ambiente:	Los casos de prueba PA_01, PA_10, PA_11, PA_12 y PA_13se ejecutaron con éxito.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Un diagrama de patrón de diseño Valido	Se muestra el mensaje de que es valido	
Observaciones			
Instancia 2	Un diagrama de patrón de diseño NO Valido	Se muestra el mensaje de que NO es valido	
Observaciones			

Tabla 42. Ejecución de prueba PA_08

Ejecución de prueba PA_09

Nombre prueba:	**PA_09**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_09		
Objetivo:	Comprobar que la herramienta es capaz de verificar combinación de patrones de diseño en el diagrama.		
Condiciones de Ambiente:	Los casos de prueba PA_01, PA_10, PA_11, PA_12 y PA_13 se ejecutaron con éxito.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	Un diagrama que contenga una combinación de patrones de diseño Valido	Se muestra el mensaje de que es valido	
Observaciones			
Instancia 2	Un diagrama que contenga una combinación de patrones de diseño NO Valido	Se muestra el mensaje de que NO es valido	
Observaciones			

Tabla 43. Ejecución de prueba PA_09

Ejecución de prueba PA_10

Nombre prueba:	**PA_10**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_10 y CU_12		
Objetivo:	Comprobar que la herramienta es capaz de crear la estructura básica de un patrón seleccionado.		
Condiciones de Ambiente:	El caso de prueba PA_01 fue ejecutado con éxito.		
Instancias de la prueba			
No. Instancia	**Entrada**	**Resultado**	
Instancia 1	El usuario selecciona el patrón de diseño “Composite”.	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Composite de	

		forma colapsada.
Observaciones		
Instancia 2	El usuario selecciona el patrón de diseño “Iterator”.	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Iterator de forma colapsada.
Observaciones		
Instancia 3	El usuario selecciona el patrón de diseño “Observer”.	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Observer de forma colapsada.
Observaciones		
Instancia 4	El usuario selecciona el patrón de diseño “Strategy”.	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy de forma colapsada.
Observaciones		
Instancia 5	El usuario selecciona el patrón de diseño “Template Method”.	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Template Method de forma colapsada.
Observaciones		
Instancia 6	El usuario selecciona el patrón de diseño “Factory Method”.	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Factory Method de forma colapsada.
Observaciones		

Tabla 44. Ejecución de prueba PA_10

Ejecución de prueba PA_11

Nombre prueba:	**PA_11**		
Ejecutor:	Manuel Bernardo Ibáñez Ocampo	**Fecha:**	11 de junio de 2015
Tipo Prueba:	Prueba de Aceptación		
Caso de uso:	CU_11 y CU_13		
Objetivo:	Comprobar que la herramienta es capaz de crear la estructura básica de un patrón seleccionado.		
Condiciones de	El caso de prueba PA_01 fue ejecutado con éxito.		

Ambiente:		
Instancias de la prueba		
No. Instancia	**Entrada**	**Resultado**
Instancia 1	El usuario selecciona la combinación de patrones de diseño "Composite-Iterator".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Composite-Iterator.
Observaciones		
Instancia 2	El usuario selecciona la combinación de patrones de diseño "Iterator-Observer".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Iterator-Observer.
Observaciones		
Instancia 3	El usuario selecciona la combinación de patrones de diseño "Observer-Composite".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Observer-Composite.
Observaciones		
Instancia 4	El usuario selecciona la combinación de patrones de diseño "Strategy-Composite".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy-Composite.
Observaciones		
Instancia 5	El usuario selecciona la combinación de patrones de diseño "Composite-Template Method".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Composite-Template Method.
Observaciones		
Instancia 6	El usuario selecciona la combinación de patrones de diseño "Composite-Factory Method".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Composite-Factory Method.
Observaciones		
Instancia 7	El usuario selecciona la combinación de patrones de diseño "Strategy-Template Method".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy-Template Method.
Observaciones		
Instancia 8	El usuario selecciona la combinación de patrones de diseño "Strategy-Factory Method".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Strategy-Factory

		Method.
Observaciones		
Instancia 9	El usuario selecciona la combinación de patrones de diseño "Template Method-Factory Method".	Se dibuja en el área de trabajo la arquitectura básica de clases para el patrón de diseño Template Method-Factory Method.
Observaciones		

Tabla 45. Ejecución de prueba PA_11

Anexo D. DISEÑO DEL INTÉRPRETE

Justificación

Las reglas hechas en la tesis (Gonzáles, 2010) y (Pérez, 2014)deben ser convertidas en archivos XML para que la herramienta SiVerPat pueda reconocerlas y usarlas en su algoritmo LL(1) (Salazar, 2013).

Convertir estas reglas manualmente resulta arduo y difícil puesto que se producen archivos de muchas líneas XML por cada patrón de diseño. Considerando que se trata de archivos XML, estos tiene etiquetas de apertura y cierre por lo cual es propenso al error humano y que el archivo no sea aceptado por SiVerPat.

El intérprete fue creado para que el usuario solo tenga que cargar el archivo TXT que contiene las reglas del patrón de diseño o combinación y sea traducido automáticamente a un archivo XML que pueda ser almacenado.

El intérprete indica si la regla tiene errores gramaticales basadas en una gramática desarrollada para generar estos archivos XML de forma correcta.

Diagrama de casos de uso

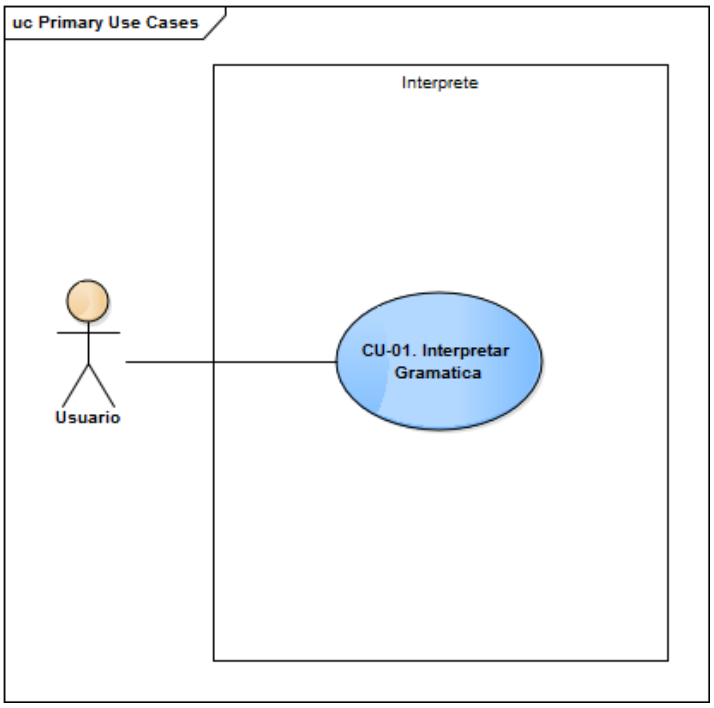

Figura 61. Diagrama de casos de uso

Escenarios

ID	**CU-01**		
Nombre del caso de uso	**Abrir TXT**		
Autor	Manuel Bernardo Ibáñez Ocampo		
Fecha de creación	12/02/2015	Fecha de Ultima modificación:	12/02/2015
Actores	Usuario		
Descripción:	El usuario carga un archivo de tipo txt.		
Precondiciones:	**Ninguna**		
Poscondiciones:	Ninguna		
Escenario principal de éxito	5 El usuario inicia el programa. 6 El usuario da clic en el botón ”Abrir”. 7 El usuario selecciona un archivo TXT. (EF1) 8 El programa carga el archivo. (EF2)		
Escenario de Fracaso(EF1)	3.1 Si el usuario no selecciona un archivo.		
Escenario de fracaso (EF2)	4.1 Si se carga el archivo con errores.		
Prioridad	Alta		
Suposiciones	**Se supone que el archivo que se carga contiene las reglas del patrón de diseño a convertir en XML.**		

Tabla 46. Caso de Uso 01

ID	**CU-02**		
Nombre del caso de uso	**Interpretar Gramática**		
Autor	Manuel Bernardo Ibáñez Ocampo		
Fecha de creación	12/02/2015	Fecha de Ultima modificación:	12/02/2015
Actores	Usuario		
Descripción:	El usuario carga un archivo de tipo txt que contiene las reglas del patrón de diseño, el intérprete lo convierte en un archivo XML para la herramienta SiVerPat.		
Precondiciones:	**CU-01**		
Poscondiciones:	CU-03		
Escenario principal de éxito	1. El usuario da clic en traducir. 2. El intérprete envía los datos para que sean analizados.(EF1) 3. El intérprete almacena la el archivo TXT, en variables locales. 4. El intérprete inicia el proceso donde crea el archivo XML 5. El sistema muestra un mensaje de que concluyo el proceso.		
Escenario de Fracaso(EF1)	2.1 Si el analizador (parser) encuentra que el archivo no está bien estructurado, emitirá un mensaje de error.		
Prioridad	Alta		
Suposiciones	**Se supone que el archivo que se carga contiene las reglas del patrón de diseño a convertir en XML.**		

Tabla 47. Caso de uso 02

ID	**CU-03**		
Nombre del caso de uso	**Guardar XML**		
Autor	Manuel Bernardo Ibáñez Ocampo		
Fecha de creación	12/02/2015	Fecha de Ultima modificación:	12/02/2015
Actores	Usuario		
Descripción:	El archivo XML es guardado en una localización que seleccione el usuario		
Precondiciones:	**CU-02**		
Poscondiciones:	Ninguna		

Escenario principal de éxito	1. El intérprete muestra la ventana de guardar cuando termine la traducción. 2. El usuario selecciona una carpeta donde será guardada, y un nombre. (EF1) 3. El intérprete guarda el archivo en el lugar especificado. (EF2)
Escenario de Fracaso(EF1)	2.1 Si el nombre del archivo ya existe, manda un mensaje si se desea sobre escribir.
Escenario de Fracaso(EF2)	3.1 Si no hay espacio, o existe algún error al guardar el archivo, se mostrara un mensaje de error.
Prioridad	Alta
Suposiciones	**Se supone que el archivo que se carga contiene las reglas del patrón de diseño a convertir en XML.**

Tabla 48. Caso de uso 03

Diseño

Diagrama de despliegue

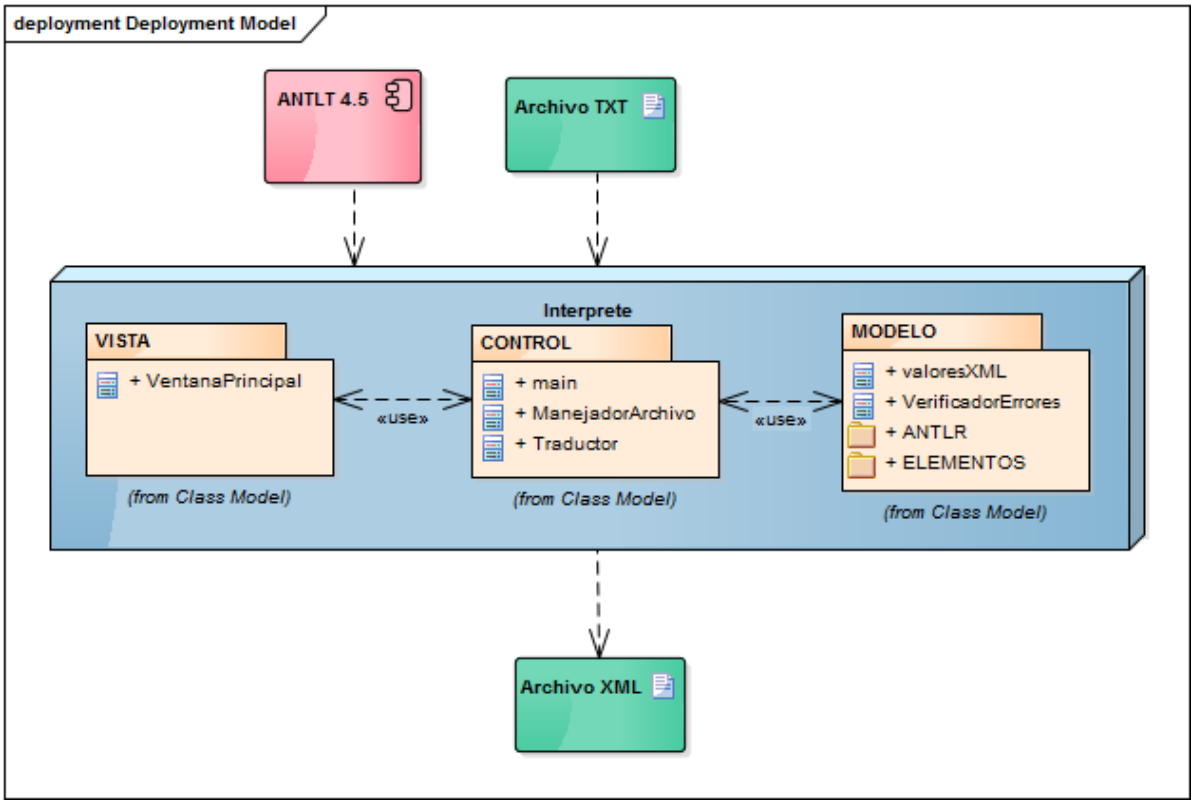

Figura 62. Diagrama de despliegue

Diagrama de paquetes

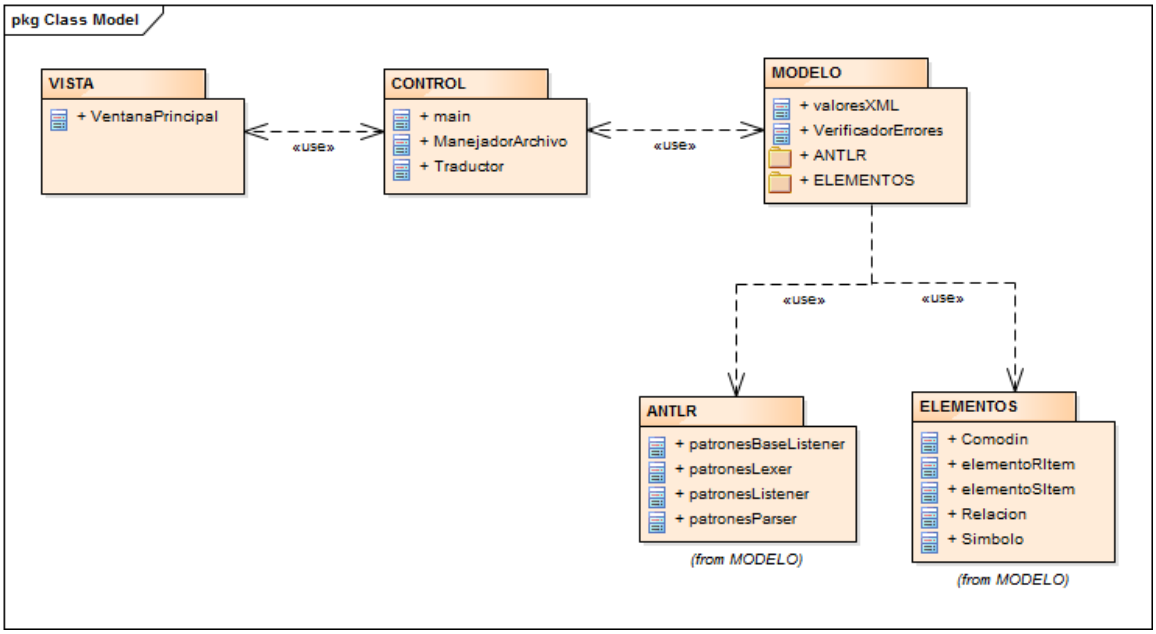

Figura 63. Diagrama de paquetes

Descripción de los Paquetes

Nombre del paquete	Descripción
VISTA	Contiene la interfaz del usuario que consiste en una clase, con una plantilla ligada.
CONTROL	Es el mediador entra la Vista y el Modelo, recibe las acciones del usuario y las dirige hacia las partes del modelo que la necesite.
MODELO	Contiene la lógica del intérprete, desde los elementos que son obtenidos del archivo TXT, la detección de errores y el proceso de traducción.
ANTLR	Contiene las clases generadas por la librería ANTLR 4.5 con la gramática patrones g4, que son usadas para obtener los valores del archivo TXT.
ELEMENTOS	Contiene las clases de los elementos que se obtienen de archivo TXT.

Tabla 49. Descripción de Paquetes

Paquete Vista

Este paquete incluye la clase que muestra la interfaz del usuario a través de la clase VentanaPrincipal.java, que tiene vinculada un archivo .form, que es un archivo con formato XML, con extensión form que es usado por la herramienta Intellij IDEA que almacena el contenido de la interfaz y su posición.

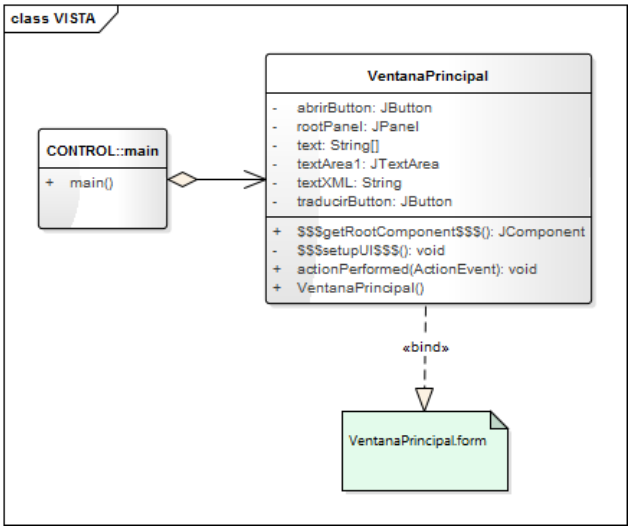

Figura 64. Paquete VISTA

Nombre:	**Main**		
Paquete:	Control		
Responsabilidad:	Clase principal, muestra la ventana del interprete.		
Capa del patrón MVC:	Control		
Atributos	**Tipo**	**Descripción**	
NA	NA	NA	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
main()	Es el cliente principal que manda a llamar el resto de la aplicación.	NA	NA

Tabla 50. Clase main

Nombre:	**VentanaPrincipal**
Paquete:	Vista

Responsabilidad:	Clase que crea la interfaz del intérprete.		
Capa del patrón MVC:	Vista		
Atributos	**Tipo**	**Descripción**	
abrirButton	JButton	Botón abrir	
rootPanel	JPanel	Panel al que todos los componentes están asignados	
Text	String[]	Variable donde se almacena el contenido del archivo TXT	
textArea1	JTextArea	TextArea donde aparecen los mensajes del interprete	
textXML	String	Variable donde se almacena la traducción en formato RAW XML	
traducirButton	JButton	Botón que inicia el proceso de traducción cuando archivo está cargado.	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
$$$getRootComponent $$$()	Clase generada que devuelve el contenido de RootPanel	NA	JComponent
$$$SetupUI$$$()	Clase generada que añade todos los componentes al RootPanel	NA	NA
actionPerformed()	Clase que recibe un evento de un componente y realiza la acción correspondiente	ActionEvent	NA
VentanaPrincipal()	Es el constructor de esta clase, carga los componentes, y añade las acciones a los componentes	NA	NA

Tabla 51. Clase VentanaPrincipal

Paquete CONTROL

El paquete CONTROL es el intermediario entre el MODELO y la VISTA, la clase manejadorArchivo es capaz de cargar un archivo y devolver su contendió y las características del archivo. La clase Traductor recibe el contenido del archivo y usando el paquete MODELO traduce el archivo en XML, el cual devuelve a la VentanaPrincipal para que el usuario seleccione el lugar para almacenarlo usando nuevamente la clase manejadorArchivo.

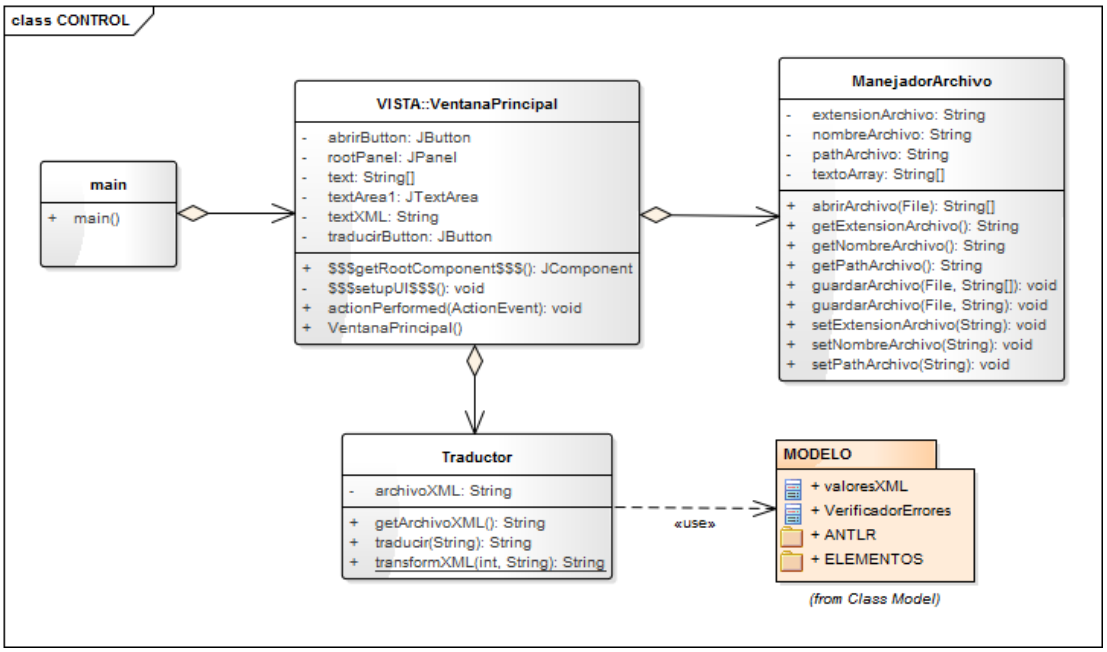

Figura 65. Paquete CONTROL

Nombre:	**ManejadorArchivo**		
Paquete:	Control		
Responsabilidad:	Clase que carga un archivo TXT y devuelve su contenido.		
Capa del patrón MVC:	Control		
Atributos	**Tipo**	**Descripción**	
extensionArchivo	String	Almacena la extensión del archivo	
nombreArchivo	String	Almacena el nombre del archivo	
pathArchivo	String	Almacena la dirección absoluta del archivo	
textoArray	String[]	Almacena el contenido del archivo.	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
abrirArchivo()	Metodo que dado un archivo lo abre y lo devuelve en un array de String	File	String[]
guardarArchivo()	Método que guarda el contenido de un array de String en el archivo especificado	File, String[]	NA
guardarArchivo()	Método que guarda el contenido de un String en el archivo especificado	File, String	NA

Tabla 52. Descripción de ManejadorArchivo

Nombre:	**Traductor**		
Paquete:	Control		
Responsabilidad:	Clase que recibe un array de String, y llama a las clases necesarias para traducirlo en un archivo XML RAW		
Capa del patrón MVC:	Control		
Atributos	**Tipo**	**Descripción**	
archivoXML	String	Almacena el resultado de la traducción en forma RAW XML	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
traducir()	Metodo que recibe una cadena con el contenido del archivo TXT y lo devuelve en XML	String	String
transformXML()	Método que recibe un String en forma RAW XML y un int con la sangría y lo convierte en un formato XML ordenado	int, String	String

Tabla 53. Descripción de Traductor

Paquete MODELO

Este paquete contiene la lógica de la herramienta, la clase verificador de errores notificara si hay algún error con la gramática, mientras que la clase valoresXML convierte el contenido del archivo TXT en XML RAW, primero almacenando los valores en diferentes variables y luego usando esas variables para construir el XML, al igual que en los paquetes anteriores, se describirá los atributos importas, y se omitirán también los métodos GET y SET de estos. Además implementa los métodos de la clase patronesBaseListener, que son de la forma enterNode o exitNode, e indican que cuando se entre o salga de algún nodo en la gramática se realizará una acción correspondiente.

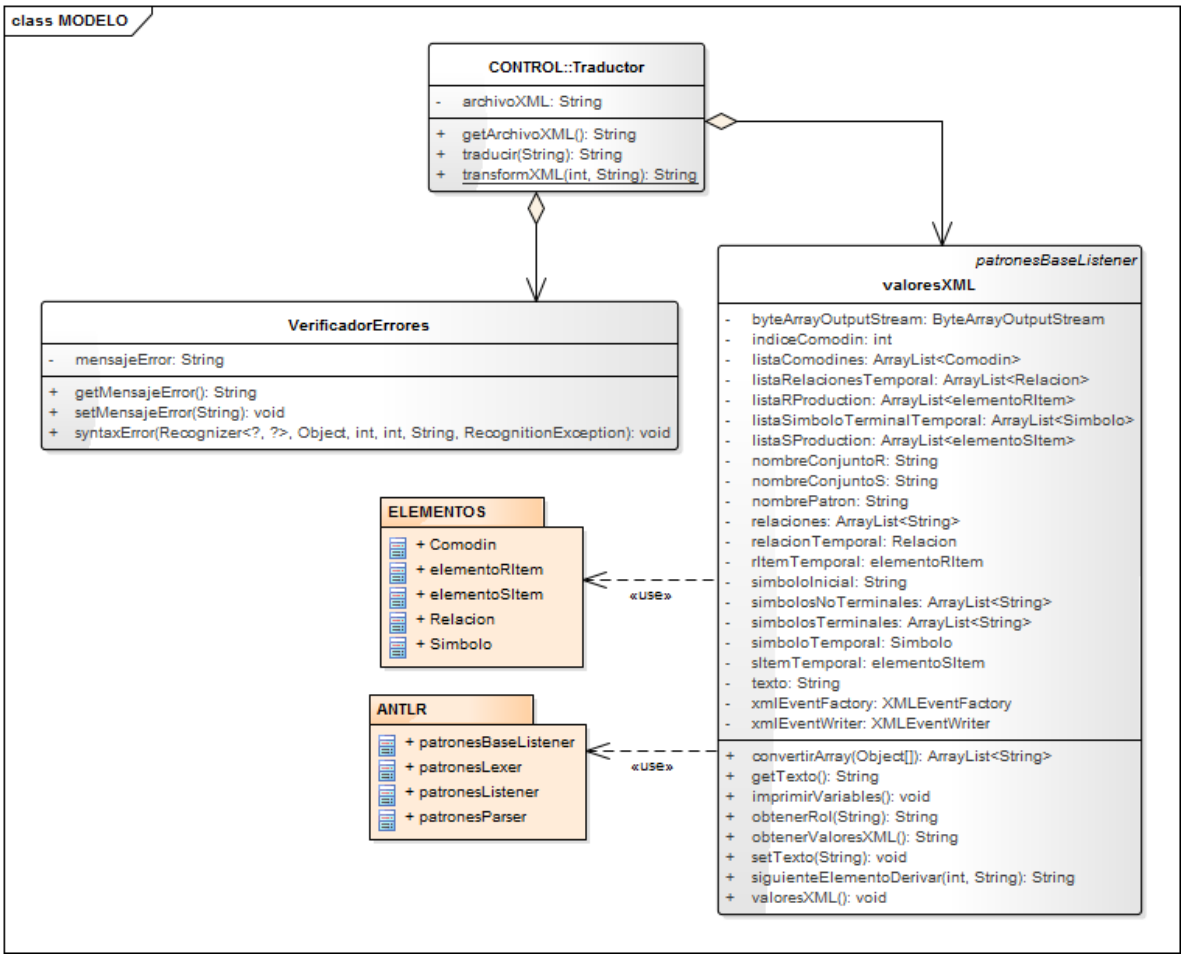

Figura 66. Paquete MODELO

Nombre:	**VerificadorErrores**		
Paquete:	MODELO		
Responsabilidad:	Clase encuentra errores en la gramática del archivo TXT y devuelve mensaje correspondiente.		
Capa del patrón MVC:	Modelo		
Atributos	**Tipo**	**Descripción**	
mensajeError	String	Almacena el mensaje de error a devolver, si no encuentra nada devuelve "vacio"	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
syntaxError()	Método que recibe un Recognizer, el símbolo que comete el error, un int con la línea, el índice de la posición del carácter en la	Recognizer<?, ?> ,Object ,int , int, String,RecognitionException	NA

	línea, el mensaje correspondiente, y la excepción.		

Tabla 54. Descripción VerificadorErrores

Nombre:	**ValoresXML**		
Paquete:	MODELO		
Responsabilidad:	Clase que recibe el contenido del archivo TXT, separa su contenido en los elementos que contiene, y convierte en XML RAW		
Capa del patrón MVC:	Modelo		
Atributos	**Tipo**	**Descripción**	
byteArrayOutputStream	ByteArrayOutputStream	Un stream donde se almacenan los bytes de salida.	
xmlEventWriter	XMLEventWriter	Objeto que almacena el conjunto de etiquetas y valores en el orden dado.	
xmlEventFactory	XMLEventFactory	Interfaz que proporciona la creación de etiquetas de forma XML.	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
convertirArray()	Método que recibe un Objeto y si es posible, lo convierte en un ArrayList de tipo String.	Object	ArrayList<String>
imprimirVaribales()	Imprime en consola el contenido de todas las variables usadas. Este método fue usado para corroborar que todas las variables necesarias fueran llenadas de forma correcta.	NA	NA
obtenerRol()	Método que recibe un símbolo NO terminal y devuelve su ROL correspondiente	String	String
obtenerValoresXML()	Método que utiliza las variables almacenadas para construir el archivo XML RAW y lo devuelve en un String	NA	String
siguienteElementoDerivar()	Método usado en el caso de que se necesite derivar el elemento siguiente al actual,	Int, String	String

	devolviendo su ROL o VACIO de no encontrarlo		
valoresXML()	Constructor de la clase	NA	NA

Tabla 55. Descripción ValoresXML

Paquete Elementos

Este paquete contiene los elementos del que está compuesta la gramática, y en donde se van a almacenar sus componentes de forma individual.

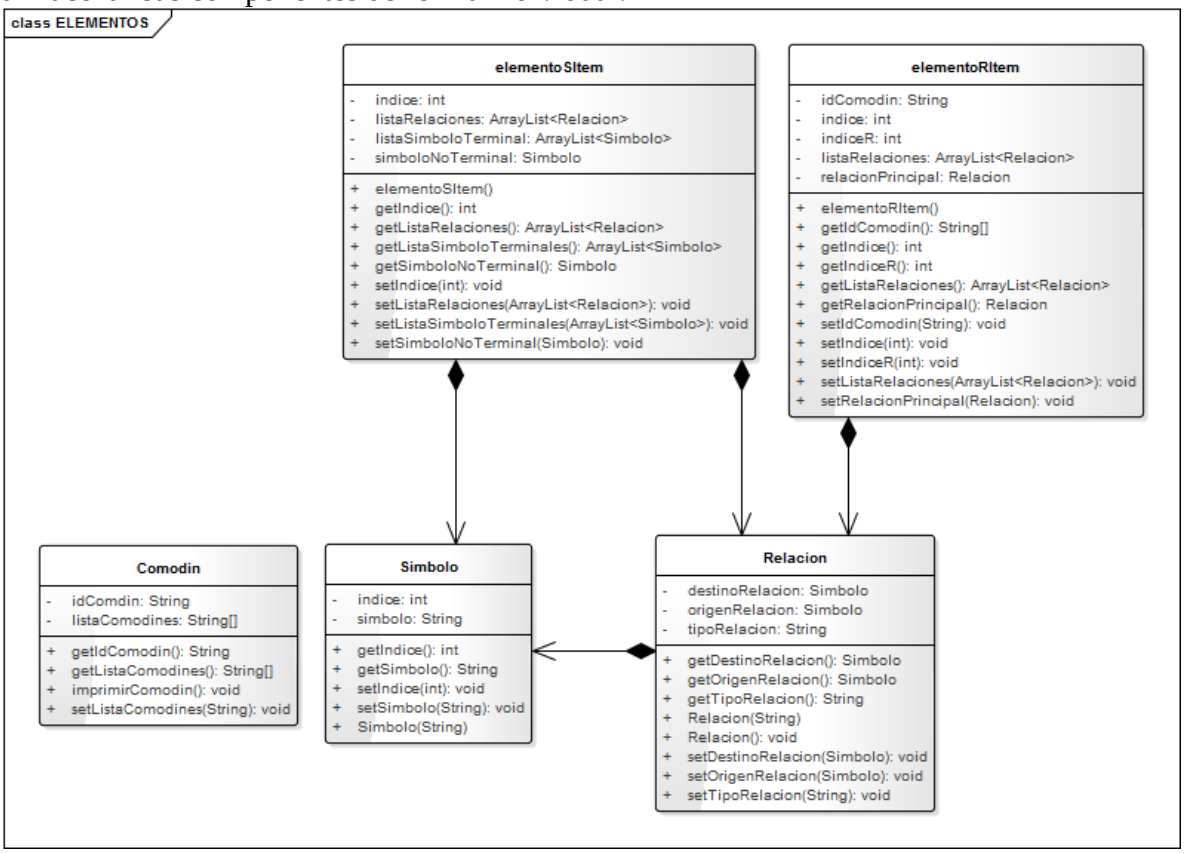

Figura 67. Paquete ELEMENTOS

Nombre:	**elementoSItem**	
Paquete:	ELEMENTOS	
Responsabilidad:	Clase que representa a un S Item	
Capa del patrón MVC:	Modelo	
Atributos	**Tipo**	**Descripción**
Índice	Int	Almacena el índice del S Item
listaRelaciones	ArrayList<Relacion>	Almacena la lista de las relaciones que

		contiene un S Item	
listaSimboloTerminal	ArrayList<Simbolo>	Almacena la lista de Símbolos Terminales de un S Item	
simboloNoTerminal	Simbolo	Almacena el Símbolo No Terminal del S Item.	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
elementoSItem ()	Constructor de la clase	NA	NA

Tabla 56. Descripción elementoSItem

Nombre:	**elementoRItem**		
Paquete:	ELEMENTOS		
Responsabilidad:	Clase que representa a un R Item		
Capa del patrón MVC:	Modelo		
Atributos	**Tipo**	**Descripción**	
idComodin	String	Almacena la id del Comodin que lo representa.	
Índice	int	Almacena el índice del R Item	
indiceR	int	Almacena el índice de la relación del R Item.	
listaRelaciones	ArrayList<Relacion>	Almacena la lista de las relaciones que contiene el R Item	
relacionPrincipal	Relacion	Almacena la relación principal del R Item	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
elementoSItem ()	Constructor de la clase	NA	NA

Tabla 57. Descripcion elementoRItem

Nombre:	**Comodin**
Paquete:	ELEMENTOS
Responsabilidad:	Clase que representa un Comodín
Capa del patrón MVC:	Modelo

Atributos	Tipo	Descripción	
idComodin	String	Almacena la id del Comodín.	
listaComodines	String[]	Almacena la lista de Comodines asociados.	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
Comodin ()	Constructor de la clase	NA	NA

Tabla 58. Descripción Comodin

Nombre:	Simbolo		
Paquete:	ELEMENTOS		
Responsabilidad:	Clase que representa un Símbolo		
Capa del patrón MVC:	Modelo		
Atributos	**Tipo**	**Descripción**	
Índice	int	Índice del símbolo.	
Símbolo	String	Almacena el valor del símbolo.	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
Simbolo ()	Constructor de la clase	String	NA

Tabla 59. Descripción Simbolo

Nombre:	Relacion		
Paquete:	ELEMENTOS		
Responsabilidad:	Clase que representa una Relación		
Capa del patrón MVC:	Modelo		
Atributos	**Tipo**	**Descripción**	
destinoRelacion	Simbolo	Almacena el destino de una relación	
origenRelacion	Simbolo	Almacena el origen de una relación	
tipoRelacion	String	Almacena el tipo de relación.	
Métodos	**Descripción**	**Datos de entrada**	**Datos de salida**
Relacion ()	Constructor de la clase	String	NA
Relacion ()	Constructor de la clase	NA	NA

Tabla 60. Descripción Relacion

Paquete ANTLR

Este paquete contiene la gramática que es usada por el intérprete para definir si el archivo TXT está correctamente redactado, llamado **patrones.g4**. Con esta gramática la herramienta ANTLR genera el siguiente conjunto de clases y archivos que permiten verificar la construcción gramatical de este archivo.

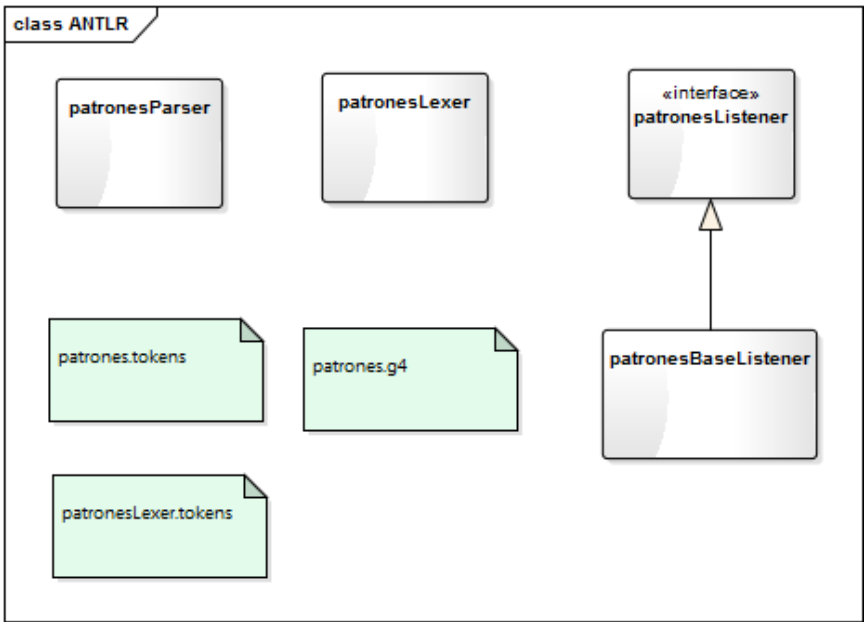

Figura 68. Paquete ANTLR

Nombre	patrones.g4
Descripción	Archivo que contiene la estructura gramática que debe tener el archivo TXT cargado.

Tabla 61. Descripción patrones.g4

Nombre	patrones.tokens
Descripción	Archivo que contiene los tokens que detecta la gramática y les asigna un valor

Tabla 62. Descripción patrones.tokens

Nombre	patronesBaseListener.java
Descripción	Clase que contiene implementaciones vacías de patronesListener.java, esta clase puede ser extendida para solo implementar un pequeño conjunto de métodos que sean necesarios.

Tabla 63. Descripción patronesBaseListener.java

Nombre	patronesLexer.java
Descripción	Clase que recibe una entrada y la convierte en conjunto de tokens.

Tabla 64. Descripción patrones.lexer

Nombre	patronesLexer.tokens
Descripción	Archivo de tokens generado por la clase patronesLexer.java

Tabla 65. Descripción patronesLexer.tokens

Nombre	patronesListener.java
Descripción	Interfaz que define todos los métodos enterNodo y exitNodo para que el parse tree pueda ser producido.

Tabla 66. Descripción patronesListener.java

Nombre	patronesParser.java
Descripción	Clase que recibe un stream de tokens, y se inicia un recorrido que verifica si el archivo está construido de forma correcta, durante este recorrido, al entrar o salir de un nodo se pueden realizar acciones específicas.

Tabla 67. Descripción patronesParser.java