PA161-3-FV-GAIA2

FRAMEWORK PARA EL DESARROLLO DE SINTAXIS CONCRETAS BASADO EN INTERFACES DE USUARIO AUMENTABLES

Federico Rodríguez Bravo

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
2016

PA161-3-FV-GAIA2

FRAMEWORK PARA EL DESARROLLO DE SINTAXIS CONCRETAS BASADO EN INTERFACES DE USUARIO AUMENTABLES

Autor:

Federico Rodríguez Bravo

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TITULO DE

MAGÍSTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Director

Jaime Andrés Pavlich Mariscal

Comité de Evaluación del Trabajo de Grado

Fernando de la Rosa

César Collazos

Página web del Trabajo de Grado

http://pegasus.javeriana.edu.co/~PA161-3-FV-GAIA2/

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERIA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Mayo 2016

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Rector Magnífico

Jorge Humberto Peláez, S.J.

Decano Facultad de Ingeniería

Ingeniero Luis David Prieto Martínez

Director Maestría en Ingeniería de Sistemas y Computación

Ingeniera Ángela Carrillo Ramos

Director Departamento de Ingeniería de Sistemas

Ingeniero Rafael González Rivera

En esta hoja solía venir la nota de aceptación del director y la firma de los jurados. Ahora, el director puede aprobar a través de correo electrónico. Esta hoja puede ser obviada.

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

Agradezco profundamente a cada una de las personas que estuvieron a mi lado en el desarrollo y la realización de este trabajo de grado, en especial a mi familia, a mi hermano y a mi director del trabajo de grado. Su compañía y apoyo fue clave para para que día a día se construyera este trabajo de grado y no me cansara de luchar por lograr algo novedoso e increíble. Muchas gracias a todos.

Contenido

ABSTRACT

CASE tools are an important support in the tasks of analysis and design of a computer software system, to visually define software models. However, the paradigm of visualization and navigation models in CASE tools has not changed significantly in recent years that means an opportunity to improve it. This work presents the conceptualization, design and implementation of a framework that allows the persistence of concrete syntax and specific template instantiation of concrete syntax applied to software models manipulated from zoomable user interfaces.

RESUMEN

Las herramientas CASE son un apoyo importante en las tareas de análisis y diseño de un sistema informático, ya que permiten definir visualmente los modelos de software. Sin embargo, el paradigma de visualización y navegación de modelos en herramientas CASE no ha cambiado significativamente en los últimos años, lo cual presenta oportunidades de mejora en la visualización, construcción de modelos y en cambios de paradigmas. Este trabajo de presenta la conceptualización, diseño e implementación de un framework que permite la persistencia de sintaxis concretas e instanciación de plantillas de sintaxis concretas aplicadas a modelos de software manipulados desde una interfaz de usuario aumentable.

RESUMEN EJECUTIVO

Este trabajo está enmarcado en el área de las herramientas CASE enfocadas en el análisis y diseño de software [1] [2]. Éstas herramientas son un conjunto de programas que incluyen capacidades automatizadas de diagramación, análisis y modelado, y están dedicadas en asistir un proyecto de software en su ciclo de vida [1] [2].

Teniendo en cuenta el contexto anterior, este trabajo de grado se realiza con el propósito de proponer una alternativa de mejora a los problemas identificados en las herramientas CASE como son: permitir qué los diagramas de diferentes tipos estén en un mismo documento y mejorar la navegación a través de los modelos y meta-modelos. De igual forma, cambiar fácilmente entre diferentes niveles de abstracción y, finalmente, presentar en los modelos sólo aquélla información que es relevante de acuerdo al nivel de detalle [3] [4]. Esta alternativa de mejora se basa en utilizar y aplicar los conceptos de Interfaces de Usuario Aumentables (ZUI), Zoom Semántico y Sintaxis Concretas [3] [5] [6], que se detallan a continuación.

- La Zoomable User Interface (ZUI) o Interface de Usuario Aumentables es una técnica de visualización donde se emplea el zoom como mecanismo primario de navegación entre diferentes niveles de abstracción. Utiliza el panning para navegar entre entidades a un mismo nivel de abstracción, y usa el zoom semántico para mostrar u ocultar información relevante dependiendo del nivel de detalle [3].

- Zoom Semántico se refiere a la forma de visualizar un objeto en una interfaz aumentable, dependiendo el nivel de zoom éste puede cambiar su representación [5] [6].

- Sintaxis Concretas: son una forma de especificar cómo representar visualmente los modelos a través de diagramas. Consisten en un conjunto de plantillas, donde cada una especifica la representación visual de cada clase del meta-modelo [5] [6].

Basado en los conceptos anteriormente descritos y en las herramientas CASE, este trabajo de grado presenta y describe todo el proceso de ingeniería de software que se llevó acabo para implementar un framework para el desarrollo de sintaxis concretas en interfaces aumentables. En otras palabras, se entiende como un conjunto de librerías y funcionalidades en una estructura tecnología compuesta de artefactos, componentes y piezas de código fuente que los desarrolladores de software utilizan para la creación de elementos de diagramas y diagramas en general, en interfaces de usuario aumentables.

Este framework se llama FV-GAIA 2.0 y provee servicios a una herramienta CASE de visualización, manipulación y modificación de elementos y diagramas de software por medio de una interfaz gráfica de usuario, como también comprende todas las estructuras de datos, objetos y funciones para la construcción de las mismas. Su desarrollo y construcción es una extensión a las funcionalidades de FV-GAIA [7], un prototipo de software que usó Interfaces de Usuario Aumentables para la creación, edición, navegación y visualización de modelos de software con buenas prácticas de Ingeniería de software. FV-GAIA surgió de la identificación de las deficiencias que presentan las Herramientas CASE y la prueba de concepto ZooMEnv [3] [7].

La oportunidad en este problema de investigación además de la extensión del prototipo FV-GAIA [7], no solo es dejar un framework que preste servicios de instanciación, manipulación y visualización de sintaxis concretas en un entorno de Interfaces Aumentables y Zoom Semántico, sino también proponer una ramificación del Documento Diagram Definition de OMG [8], aplicado a el entorno anterior mente mencionado. Por lo tanto, este trabajo de grado comprende todas las especificaciones y diseños de software enfocados en la construcción de FV-GAIA 2.0 junto con sus entregables como lo son: Documentos Visión, SRS, SDD, reportes de pruebas y manuales de usuario, como también FV-GAIA 2.0 como producto de software.

Como resultados y productos de este trabajo de grado, además de cada uno de los entregables que acompañan al producto de software FV-GAIA2.0, se realizó uno editor de sintaxis concretas que le permite a un usuario crear plantillas de sintaxis concretas a partir de ciertas primitivas gráficas por medio de una interfaz gráfica, con el propósito de crear elementos de diagramas de acuerdo a las necesidades del usuario y a partir de esto, generar meta-modelos de software y lenguajes de modelado propios. En este editor, las primitivas gráficas se podrán mover, girar, escalar, editar, agrupar, contener dentro de otras, acomodar de forma arbitraria, agregar distintas presentaciones y layouts para luego ser guardadas como plantillas de elementos de diagramas.

Otro resultado de este trabajo de grado es el framework de estructuras de datos que ofrece a los desarrolladores de software todas las funcionalidades de un conjunto de librerías y métodos para la instanciación y generación de sintaxis concretas en interfaces de usuario aumentables. El último resultado de este trabajo de grado es la extensión del documento Diagram Definition de OMG [8], donde se tomaron los modelos originales de este documento y se aplicaron a las necesidades de generar sintaxis concretas en interfaces aumentables lo que generó una nueva versión de estos modelos.

Como conclusiones de este trabajo de grado, se enumeran las siguientes:

- FV-GAIA 2.0 es un Framework que cumple con las expectativas propuestas en términos funcionales y no funcionales, ya que está implementado con buenas prácticas de ingeniería de software, lo que permite que sea usable y se puedan realizar actualizaciones sobre este. La curva de aprendizaje para desarrollar sobre FV-GAIA 2.0 no es muy costosa en términos de tiempo.
- La generación de sintaxis concretas en interfaces aumentables es posible por medio de una interfaz de usuario, como por medio de código fuente.
- FV-GAIA 2.0 es un framework más robusto, flexible y escalable en comparación a FV-GAIA, ya que no solo tiene muchas más funcionalidades sino que también se puede aplicar para desarrollar cualquier tipo de diagrama visual.

INTRODUCCIÓN

El presente trabajo está enmarcado en el área de las herramientas CASE enfocadas en el análisis y diseño de software. Éstas herramientas son un conjunto de programas que incluyen capacidades automatizadas de diagramación, análisis y modelado, y están dedicadas en asistir un proyecto de software en su ciclo de vida [1] [2].

Así como las herramientas CASE asisten de forma correcta a los ingenieros de sistemas en el diseño y desarrollo de software, también presentan varias oportunidades de mejora. En primera instancia, permitir qué los diagramas de diferentes tipos estén en un mismo documento y mejorar la navegación a través de los modelos y meta-modelos. De igual forma, cambiar fácilmente entre diferentes niveles de abstracción y, finalmente, presentar en los modelos sólo aquélla información que es relevante de acuerdo al nivel de detalle [3] [4].

Una de las formas más adecuadas de mejorar, extender y potenciar las herramientas CASE es haciendo uso de Interfaces de Usuario Aumentables (ZUI), Zoom Semántico y Sintaxis Concretas [3] [5] [6].

Basado en lo anterior, se realizó un trabajo de grado por el mismo autor del trabajo de grado actual, para el pregrado de Ingeniería de Sistemas de la Pontificia Universidad Javeriana. Este trabajo consistió en elaborar un prototipo llamado FV-GAIA, el cuál usó Interfaces de Usuario Aumentables para la creación, edición, navegación y visualización de modelos de software con buenas prácticas de Ingeniería de software. FV-GAIA surgió de la identificación de las deficiencias que presentan las Herramientas CASE y la prueba de concepto ZooMEnv [3] [7].

ZoomEnv se describen en [3] et al como una herramienta donde se extienden los conceptos de sintaxis concretas de lenguajes visuales a través de un meta-modelo complejo, a sintaxis concretas visuales en interfaces de usuario aumentables. En la ilustración 1 se muestra la arquitectura que caracteriza esta herramienta. En el marco teórico se explica esta herramienta en detalle [3].

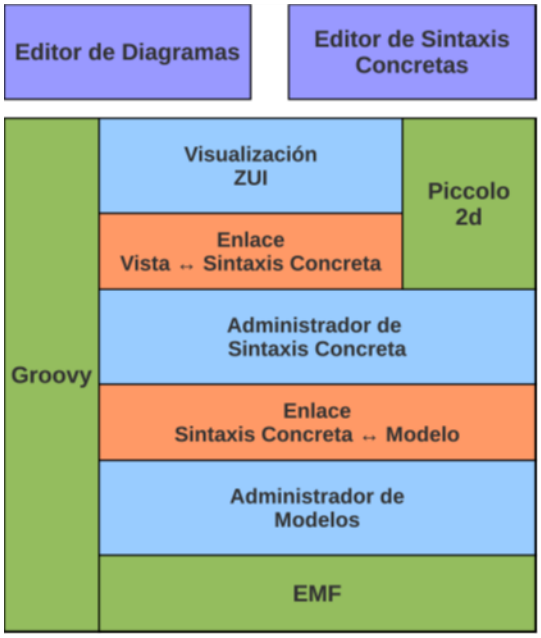


Ilustración 1: Arquitectura de software ZooMEnv [3]

En este trabajo de grado se realiza una extensión a las funcionalidades de FV-GAIA, permitiendo que el usuario no solo pueda visualizar diagramas de software en interfaces aumentables sino también pueda modelar, configurar y diseñar sus propios elementos de Sintaxis Concretas y lenguajes de modelado según sus necesidades por medio de un editor visual, como también por medio de código fuente ya que este es un framework que presta servicios de creación e instanciación de sintaxis concretas en interfaces aumentables.

Adicional a esta extensión de software, el diseño y la arquitectura de software de las capas relacionadas con la administración de Sintaxis Concretas, están basadas en el Documento Diagram Definition de OMG [8], que no solo se aplican los diferentes modelos de software en FV-GAIA 2.0, sino que se propone una ramificación de este documento con de los mismos modelos de software aplicados a Interfaces de Usuario Aumentables y Zoom Semántico.

El presente documento tiene la siguiente estructura: en la primera y segunda sección se presenta la descripción general y la descripción del proyecto del trabajo de grado realizado, en la tercera sección se presenta el marco teórico, en la cuarta sección se presentan los trabajos relacionados al trabajado de grado presentado, y por último en la quinta sección se detalla cada uno de los entregables y productos realizados durante este trabajo de grado.

1. DESCRIPCIÓN GENERAL

Oportunidad y Problemática

El objetivo de este trabajo de grado es extender la versión actual de FV-GAIA con componentes nuevos que permitan administrar sintaxis concretas de forma visual para lograr que el

usuario pueda configurar y diseñar sus propios elementos de diagramas de acuerdo a sus necesidades. Como también editar y persistir sintaxis concretas sin necesidad de quemarlas a nivel de código fuente como era el caso de FV-GAIA [7].

El desarrollo de FV-GAIA 2.0 se basó en FV-GAIA [7], este es un framework que permitió probar si era posible la visualización de modelos de software en interfaces de usuario aumentable. Este framework provee todos los bloques de construcción de meta-modelos gráficos que permite no solo visualizar elementos y diagramas de software en una interfaz aumentable, sino también ofrece servicios de instanciación de objetos y llamar funciones para visualizar grafos en este tipo de interfaces [7].

FV-GAIA 2.0 es un framework que provee servicios de visualización, manipulación y modificación de elementos y diagramas de software por medio de una interfaz gráfica a una herramienta CASE. Esta interfaz está compuesta por dos editores visuales: un editor de sintaxis concretas y un editor de diagramas. El primero permite crear plantillas de elementos de diagramas de acuerdo a unas primitivas gráficas establecidas. El segundo es un editor de diagramas donde se utilizan las plantillas de elementos de diagramas creadas por el usuario para la realización de diagramas de software. La implementación de esta nueva versión de este aplicativo se desarrolló sobre Java y la librería ZVTM. A diferencia de su primera versión, las funcionalidades y características aumentaron.

La Ilustración 2 muestra cómo es la interfaz gráfica de FV-GAIA [7], donde el usuario tiene la posibilidad de visualizar y manipular diagramas de software, pero no se le permite la modificación de los elementos de estos diagramas. Esta interfaz gráfica y funcionalidades de FV-GAIA tienen un diseño de software particular. En la Ilustración 3 se muestra la arquitectura que caracteriza a este framework. Es una arquitectura basada en el patrón Modelo-Vista-Controlador [35] para representar, diseñar y desarrollar aplicaciones con énfasis en interfaces gráficas, y encapsula sus componentes por capas [36].

Esta Arquitectura se compone de dos capas y un componente transversal para ambas capas. La primera capa es Controlador ZUI, se encarga de administrar los eventos que se presenten en la interfaz gráfica y componentes visuales asociados a nodos y aristas. La segunda capa es Visualización ZUI, esta se encarga de presentar cada nodo o arista de forma visual, representado cada uno de estos elementos visuales en estructuras de datos y objetos. La Ilustración 2 muestra cómo es la arquitectura actual de FV-GAIA [7] junto con sus componentes, de acuerdo a la utilización de la librería ZVTM.

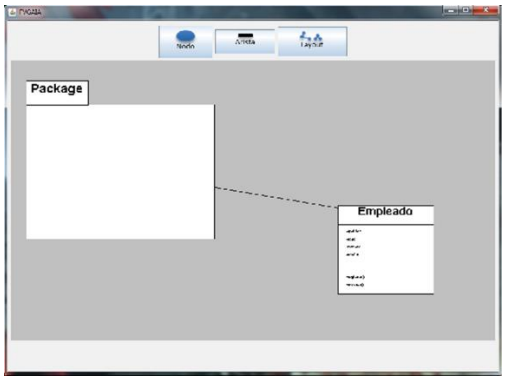


Ilustración 2: Interfaz gráfica de FV-GAIA

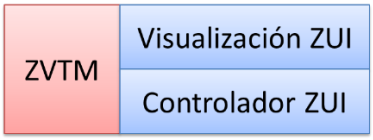


Ilustración 3: Arquitectura de Software FV-GAIA

La Ilustración 4 muestra cómo sería la interfaz gráfica de FV-GAIA 2.0. En esta imagen se puede ver la gran diferencia que existiría entre las dos versiones y se puede identificar que el editor de diagramas es similar al que había en la interfaz gráfica de FV-GAIA [7]. Pero lo novedoso en esta versión es la interfaz gráfica adicional para la creación de plantillas de elementos de diagramas, que es el editor de sintaxis concretas. Este editor da flexibilidad a la herramienta y al usuario en términos de usabilidad, como también le permite al usuario no solo crear elementos de diagramas personalizados sino definir su propio lenguaje de modelado.

La Ilustración 5 muestra cómo sería la arquitectura de software de FV-GAIA 2.0, de la cual hacen parte varios componentes. Entre estos, algunos permiten la especificación de los modelos y meta-modelos para la representación de las plantillas de elementos de diagramas, como también están los componentes de la arquitectura de FV-GAIA [7].

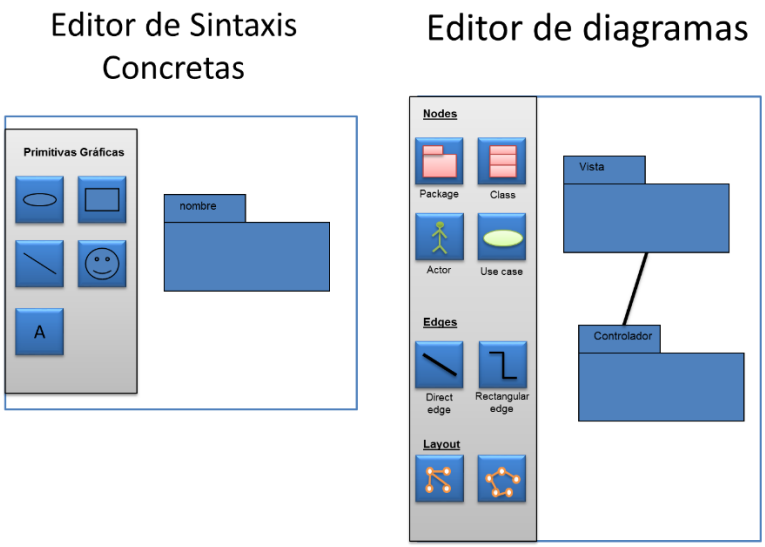


Ilustración 4: Interfaz gráfica de FV-GAIA 2.0

Ilustración 5: Arquitectura de Software FV-GAIA 2.0

Las comparaciones anteriores entre lo que es FV-GAIA [7] y lo que será FV-GAIA 2.0, muestran cómo serán las mejoras, las funcionalidades y las características que tendrá la nueva versión en comparación a la vieja versión de FV-GAIA [7]. La nueva versión de este framework se acercará al propósito de hacer el framework base de una Herramienta CASE con interfaces aumentables para apoyar el análisis, diseño y desarrollo de un producto de software en su ciclo de vida.

Debido a restricciones de tiempo y recursos, este trabajo de grado no abarca la totalidad de la arquitectura de ZooMEnv. En particular, la administración de modelos, componente requerido para crear una herramienta CASE completa, será desarrollado en un trabajo futuro.

La oportunidad en este problema de investigación además de la extensión del prototipo FV-GAIA [7], no solo es dejar un framework que preste servicios de instanciación, manipulación

y visualización de sintaxis concretas en un entorno de Interfaces Aumentables y Zoom Semántico, sino también proponer una ramificación del Documento Diagram Definition de OMG [8], aplicado a el entorno anterior mente mencionado.

2. DESCRIPCIÓN DEL PROYECTO

2.1 Objetivo general

Extender FV-GAIA para el desarrollo de sintaxis concretas basado en interfaces de usuario aumentables.

2.2 Objetivos específicos

1. Especificar los requerimientos de FV-GAIA 2.0.
2. Extender el estándar Diagram Definition de OMG para soportar interfaces aumentables y zoom semántico.
3. Especificar el diseño de FV-GAIA 2.0, basado en los requerimientos y la extensión al estándar Diagram Definition.
4. Extender las funcionalidades actuales de FV-GAIA, soportado en la especificación de diseño anterior.
5. Validar FV-GAIA 2.0 a través de pruebas de aceptación.

2.3 Fases de desarrollo

Para el análisis, diseño y desarrollo de este trabajo de grado y del framework FV-GAIA 2.0 se utilizó como metodología de desarrollo de software: Proceso Unificado Ágil (AUP), esta hace uso de técnicas ágiles para que en todas las fases sea posible mejorar la productividad del software a realizar. AUP es una versión simplificada de RUP [9].

Esta metodología aplicada al proyecto, se dividió en las siguientes fases, haciendo referencia al Proceso Unificado Ágil (AUP).

1. Concepción.
2. Elaboración.
3. Construcción.

Complementando AUP, también se utilizó la metodología Kanban durante todo el desarrollo del FV-GAIA 2.0. Kanban es una manera de gestionar el trabajo de una forma fluida. Hace uso de un tablero de tareas que permite visualizar de forma global el progreso del trabajo. También permite que cualquier persona se mantenga actualizada sobre el estado del mismo [19].

3. MARCO TEÓRICO / ESTADO DEL ARTE

Teniendo claro la problemática planteada se puede entrar a explicar los conceptos claves a considerar para el entendimiento global del proyecto que se plantea.

A continuación se describen los conceptos y trabajos relacionados más importantes para entender el propósito de la investigación y del trabajo de grado.

3.1. Conceptos Fundamentales

En primer lugar se encuentran los conceptos generales que se refieren a terminología conocida en ingeniería de sistemas, que se aplican en el presente trabajo de grado. El primer concepto se refiere a las *Herramientas CASE* que pretenden ser un apoyo en todas las etapas del ciclo de vida de un producto de software. Estas se define como: "Un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases".

Este tipo de herramientas son tantas que es pertinente clasificarlas dependiendo su funcionalidad y su aplicación en el área del desarrollo de software y se clasifican según: las plataformas que soporta, las fases del ciclo de vida del desarrollo de sistemas que cubren, la arquitectura de las aplicaciones que producen y su funcionalidad [23]. Este tipo de herramientas de software es el enfoque por el cual se construyó FV-GAIA 2.0 basado en los aspectos de mejora de estas.

Muy relacionado con las Herramientas CASE, en términos de la estructura y construcción de los elementos de diagramas se encuentran las *Sintaxis Concretas*, que son una forma de especificar cómo representar visualmente los modelos a través de los Diagramas. Consisten en un conjunto de plantillas, donde cada una especifica la representación visual de cada clase del meta-modelo [5] [6]. Para completar los conceptos relacionados con Herramientas CASE están las *Primitivas Gráficas*, que se relacionan con estas herramientas CASE en la creación gráfica de elementos. Para el presente trabajo se definen como gráficos vectoriales, como los siguientes: líneas, splines, rectángulos, óvalos, polígonos, imágenes y texto. En el contexto de la aplicación práctica se definen como Glyph que hace parte de la librería ZVTM [15] [20].

Dejando a un lado las Herramientas CASE, se encuentran las *Interfaces de Usuario Aumentables o (ZUI)*, esta es una técnica de visualización donde se emplea el zoom como mecanismo primario de navegación entre diferentes niveles de abstracción. Utiliza el panning para navegar entre entidades a un mismo nivel de abstracción y usa el *Zoom Semántico* para mostrar u ocultar información relevante dependiendo del nivel de detalle [3]. Este último se refiere a la forma de visualizar un objeto en una interfaz aumentable, dependiendo el nivel de zoom éste puede cambiar su representación [5] [6]. El *Nivel de Detalle* se refiere a la representación de elementos, este varía de acuerdo al nivel de zoom. Los elementos se esconden a bajo nivel de zoom y se muestran a alto nivel de zoom [3] [21].

En segundo lugar se encuentran los conceptos base que se refieren a los conceptos e ideales en los cuales se fundamenta o se estructura el presente trabajo de grado. De acuerdo a lo anterior, por un lado se encuentra *ZooMEnv* que es una herramienta de modelado y meta modelado que permite crear de forma efectiva, elementos de diagramas con sintaxis concretas basadas en ZUI [22]. Es un aplicativo que no satisface en su totalidad los requerimientos establecidos, ya que su módulo de visualización ZUI no es del todo eficiente y no presenta un buen desempeño de la herramienta al utilizar grafos de gran tamaño.

ZooMEnv se especificó con una arquitectura de software que facilite la extensión de su funcionalidad en el tiempo. Está acompañada por las siguientes tecnologías y librerías: Piccolo 2d [14], Groovy [24] y EMF [25], las cuales soportan diferentes componentes de la misma. La ilustración 6 presenta la arquitectura de esta herramienta, que comprende módulos principales como administrador de grafos, administrador de sintaxis concretas y Visualización ZUI [3] [22]. Esta arquitectura así como sus componentes, son la base para la especificación arquitectural de FV-GAIA 2.0. [3].

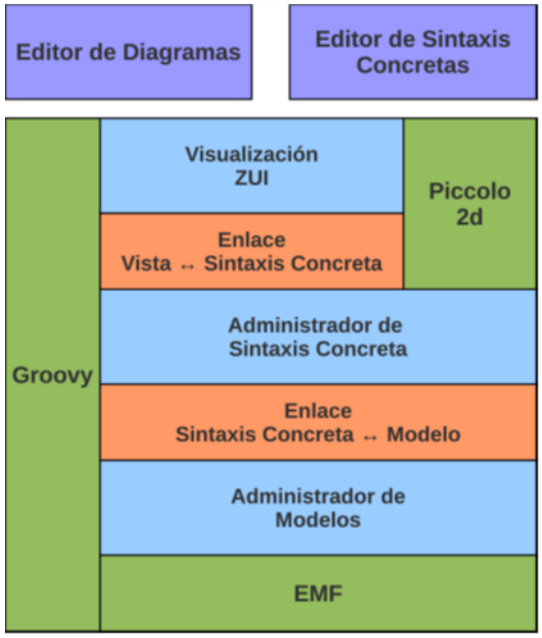


Ilustración 6: Arquitectura de software ZooMEnv [3]

Por otro lado, en los conceptos base se encuentra *Diagram Definition* el cual provee una base para la elaboración de modelos y el intercambio de notaciones gráficas. Específicamente en diagramas de nodos y aristas del estilo que se encuentran en UML, SysML, y BPMN. Donde las notaciones están vinculados a la sintaxis del lenguaje [8]. Este es una especie de framework que por medio de modelos base permite definir diagramas para la construcción de las estructuras y objetos que permitirán construir la sintaxis concreta de un elemento de diagrama.

Diagrama Definition permite: a) Definición del lenguajes específico para el intercambio de diagramas de meta modelos, y B) Mapeo de instancias de lenguajes especificados de meta modelos construidos. Este framework se basa en manejar dos tipos de información en sus modelos: el primero se refiere a la información específica de la definición del diagrama y sus especificaciones de lenguaje de modelado relacionado, y el segundo tipo de información, es información gráfica de los modelos como posición y forma [8].

Diagram Definition presenta la siguiente arquitectura basada en los tipos de información explicados con anterioridad.

- *Diagram Interchange (DI):* Permite el intercambio de información gráfica que los usuarios del lenguaje tienen control sobre ellos, tales como la posición de los nodos y puntos de enrutamiento de línea.
- *Diagram Graphic (DG):* Contiene un modelo de primitivas gráficas que se pueden crear instancias para hacer el mapa de un lenguaje de sintaxis de modelos abstractos y modelos del intercambio diagrama (DI) para presentaciones visuales
- *Diagram Common (DC):* Contiene un número de tipo de primitivas comunes como tipos de datos estructurados usados por los paquetes: "Diagram Interchange (DG)" y "Diagram Graphics (DI)"

La Ilustración 7 muestra como es la arquitectura de Diagram Definition de acuerdo a lo explicado con anterioridad.

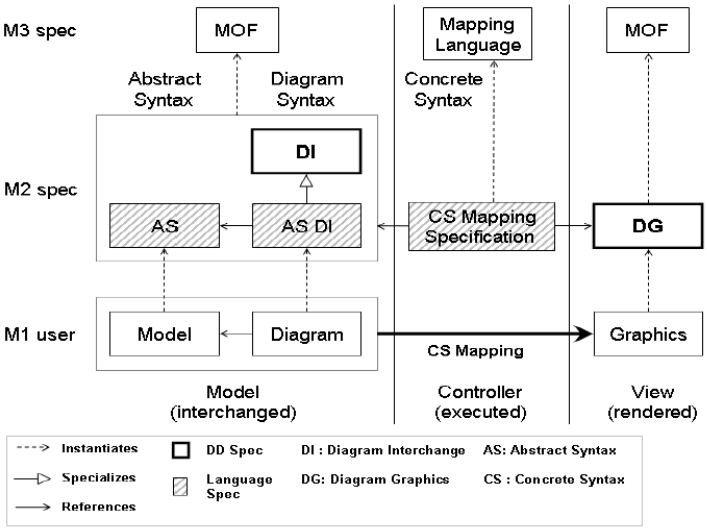


Ilustración 7: Arquitectura Diagram Definition [8]

Por último, en tercer lugar se encuentra los conceptos específicos de FV-GAIA 2.0, los cuales se refieren a la administración de plantillas y diagramas. En primer lugar está la *Plantilla de Sintaxis Concretas*, que es un elemento construido a partir de primitivas gráficas como: círculos, cuadrados, óvalos, textos o imágenes que representan un molde de un elemento de diagrama, en otras palabras es la estructura o el esqueleto para la instanciación de un elemento de diagrama. Estas plantillas se guardan en *Documentos de Plantillas* que es el espacio en memoria donde se guarda todo el diseño y la configuración de cada plantilla de sintaxis concretas, que a su vez cada uno de estos documentos hace parte de una *Plantilla*, que es una estructura que aloja varios documentos de este tipo.

Para usar estas plantillas de sintaxis concretas y poder crear elementos de diagramas y diagramas a partir de estas, existen otros conceptos adicionales que complementan los anteriormente descritos. También se definen las *Instancias de Plantilla de Sintaxis Concretas*, que

son objetos copias instanciadas de las plantillas de sintaxis concretas configuradas y diseñadas con anterioridad, la cuales se guardan en *Documento de Diagrama* que es el espacio en memoria donde se almacena cada instancia de plantilla de sintaxis concretas junto con su configuración y estructura. Cada uno de estos documentos hace parte de una estructura llamada *Diagrama*, que aloja varios documentos de este tipo.

3.2 Trabajos relacionados

De acuerdo a la problemática planteada con anterioridad, se describen los distintos trabajos de grado, artículos y aplicativos relacionados.

Algunos trabajos relacionados se han realizado en la Pontificia Universidad Javeriana a cargo de estudiantes de Pregrado y Posgrado de Ingeniería de Sistemas. Todos estos trabajos han sido dirigidos por el Ing. Jaime A. Pavlich-Mariscal, los cuales se basaron en ZooMEnv [3] y FV-GAIA [7].

El primer trabajo de grado relacionado de la referencia [42] se enfocó en realizar un producto final llamado ZoomTI++. El producto es un framework que permite la visualización de diagramas de software por medio de interfaces aumentables, está desarrollado en C++ para dispositivos multi-touch. Este framework se hizo con el propósito de probar una nueva aproximación y plataforma para la visualización de diagramas de software en niveles de abstracción diferente, teniendo en cuenta los conceptos de Interfaces de Usuario Aumentable, Zoom y Zoom Semántico.

El segundo trabajo [10], consiste en un estudio sobre el uso de interfaces de usuario aumentables (ZUI) aplicadas a la visualización de modelos de software. El objetivo es determinar si realmente esta técnica de visualización mejora su comprensión y es aceptada por parte de los usuarios. Este trabajo incluye el desarrollo de una herramienta web, llamada ZMT, que tiene en cuenta los conceptos de Interfaces de Usuario Aumentable, Zoom y Zoom Semántico aplicado a grafos.

En cuanto a artículos y papers relacionados a Interfaces de Usuario Aumentables, Zoom Semántico y visualizaciones de modelos de software en interfaces ZUI, se encuentran los siguientes trabajos:

El trabajo de Geiger et al. [11] presenta el diseño, la implementación y evaluación de una interfaz aumentable dedicada a mostrar modelos jerárquicos de gran tamaño de un complejo sistema de mecatrónica. La problemática se relaciona con interfaces aumentables, ya que las usan para presentar diagramas de más de ochocientos elementos cada uno. Sin embargo, no son diagramas como los conocidos en el área de ingeniería de sistemas, sino diagramas que junta la mecánica, electrónica, ingeniería de control e ingeniería de software. Estos diagramas no son proyectados en dispositivos comunes, sino en pantallas gigantes. Se diferencia con la problemática que se plantea en el manejo y en la creación de diagramas que es totalmente diferentes, enfocados a ingeniería de sistemas y en su presentación al usuario, que es en un computador de escritorio o portátil.

El artículo de Bennett et al. [12] describe el uso de interfaces aumentables para representar el concepto de ORRIL (Objects, Regions, Relations and Interface Logic). El objetivo principal de este es cómo funcionan las interfaces aumentables y como por medio de objetos o artefactos que se crean en una interfaz de este tipo se le pueda explicar a usuario su funcionamiento.

El trabajo de Liu et al. [13] detalla la integración de una interfaz de usuario aumentable con una herramienta meta-CASE llamada Pounamu, se utiliza para especificar y realizar una amplia gama de entornos de lenguaje visuales orientados a dominios específicos. Fue desarrollada para la construcción de lenguajes de entornos de múltiples vista por la Universidad de Auckland en Nueva Zelanda. La integración de ZUI a Pounamu proporcionó características mejoradas en navegación de vistas y gestión de las mismas. También ayudó a solucionar problemas, como la falta de espacio en la pantalla y la capacidad de visualización de complejas vistas.

En cuanto a aplicativos existen, hay varias librerías y frameworks que permiten el desarrollo de las mismas, como las descritas a continuación:

Piccolo2D: Es un capa construida en lo más alto de las API´s graficas de bajo nivel como OpenGL [40] y VRLM [41]. Es una herramienta que soporta el desarrollo de programas gráficos estructurados en 2D, particularmente el desarrollo de interfaces aumentables [14]. Usa un modelo tipo Scene Graph que es común en entornos 3D [14], mantiene una estructura jerárquica de objetos y cámaras, permitiéndole al desarrollador de aplicaciones una mejor orientación y manipulación de objetos de manera significativa, y permite crear aplicaciones graficas estructuradas sin preocuparse mucho por aspectos de bajo nivel.

ZVTM es un herramienta implementada en java, diseñado para facilitar la tarea de la creación de complejos editores visuales en los que una gran cantidad de objetos que se mostrarán. Contiene formas geométricas complejas y sus respectivas animaciones [15]. Además de esto, es capaz de manejar grandes cantidades de objetos de forma eficiente, se basa en la metáfora de los universos que se pueden observar a través de cámaras móviles inteligentes de acercamiento y de movimiento, ofrece soporte para vistas en múltiples capas y trasparencia de objetos, y tiene facilidades de utilizar animaciones en primitivas gráficas. Por último, su modelo de objetos gráficos permite que tareas como crear, modificar y animar entidades sea más fácil.

Eagle Mode: es una solución avanzada en la que el usuario puede visitar casi todo, simplemente por medio de la navegación a través del zoom [16]. Es un tipo de interfaz de usuario aumentable dedicada a mostrar una versión futurista de la interacción Humano-Computador donde el usuario puede visitar casi todo por medio del Zoom. Es una aplicación de escritorio, que convierte el escritorio del computador en una interfaz de usuario aumentable con un administrador de archivos, de vistas, juegos y más. Esta desarrollado en C++.

Clutter: es una librería Open Source para la creación de interfaces gráficas de velocidad, peso, portabilidad y de usuario, totalmente dinámicas [17]. Esta librería utiliza OpenGL [40] para renderizar, es un API fácil de usar, eficiente y flexible alrededor de la complejidad de GL. No cumple con ningún estilo de interfaz de usuario en particular, pero proporciona una

base rica y genérica para herramientas gráficas de nivel superior adaptadas a las necesidades específicas.

CZui: es un framework que permite la creación de interfaces aumentables en C++, Se ha diseñó para utilizar cualquier gráfico y biblioteca de fuentes [18]. Utiliza el concepto grafo de la escena, con la posibilidad de encuadre y zoom de cámaras en diferentes partes de la misma escena. Varias cámaras pueden ser relacionadas en la misma ventana, cada una con su rectángulo de pintura.

Complementando la información anterior, en la siguiente tabla se presenta un cuadro comparativo entre los productos de software generados en trabajos relacionados y el generado para el presente trabajo de grado donde se puede ver como FV-GAIA 2.0 satisface más características que el resto de productos de software.

	Creación de sintaxis concretas	**Creación de Diagramas**	**Manipulación de elemento**	**Visualización de grafos**	**Uso de ZUI**	**Flexibilidad y usabilidad**
FV-GAIA 2.0	✓	✓	✓	✓	✓	✓
FV-GAIA		✓	✓	✓	✓	
ZoomTI++		✓		✓	✓	✓
ZMT		✓		✓	✓	✓
Geiger				✓	✓	✓
ORRIL						
Pounamu		✓	✓	✓	✓	
Frameworks y librerías gráficas					✓	

Tabla 1: Cuadro comparativo entre FV-GAIA 2.0 y trabajos relacionados

5. DESARROLLO DEL TRABAJO DE GRADO

En las siguientes secciones se describirá el desarrollo del trabajo de grado en términos de entregables, como también se describirán los detalles más relevantes que permitieron llegar a construir FV-GAIA 2.0 como framework.

I – Documento Visión

El documento Visión fue el primer entregable del trabajo de grado, el cual pertenece a la fase de concepción de AUP [9]. Este documento permitió definir a FV-GAIA 2.0 en términos de producto de software.

a) Definición del Problema

Antes de definir FV-GAIA 2.0 como producto, se definió el problema a resolver teniendo en cuenta el aplicativo FV-GAIA [7]. El problema de FV-GAIA consistía en que la usabilidad y flexibilidad de este framework se ven afectadas definiendo sintaxis concretas por medio código fuente, ya que es necesaria la re compilación de la aplicación cuando se quiera realizar algún tipo de cambio en éstas sintaxis.

Basado en esta problemática y a las deficiencias que presentaba FV-GAIA se propuso que una solución adecuada sería extender FV-GAIA generando un módulo y componentes adicionales que permitieran mejorar la flexibilidad y usabilidad del aplicativo, lo que llevó a definir FV-GAIA 2.0 como producto de software.

b) Descripción Global del Producto

De acuerdo a la definición del problema se estableció FV-GAIA2.0 de la siguiente manera.

Un framework que presenta servicios de visualización de sintaxis concretas en una interfaz de usuario aumentable, este framework debe tener la capacidad de poder ser escalable y al mismo tiempo de integrarse con otros módulos o componentes para formar la capa de visualización de una herramienta CASE con ZUI. Adicional a esto, se estableció que FV-GAIA 2.0 no tendrá sintaxis concretas quemadas en el código fuente, se generará una nueva interfaz gráfica para la manipulación de sintaxis concretas, se modificará la interfaz de edición de diagramas, y además del concepto de sintaxis concretas, FV-GAIA 2.0 tendrá todas las funcionalidades que ya tenía FV-GAIA.

c) Definición de usuarios finales

Teniendo en cuenta la descripción y el problema se establecieron los potenciales usuarios finales los cuales serían Desarrolladores. Estos se describen como: persona o grupo de personas que hace uso de FV-GAIA 2.0 o externas que pueden ser: el director de trabajo de Grado, un grupo de estudiantes de ingeniería de sistemas o desarrolladores de software. Estos tienen responsabilidades como: explorar todas las funcionalidades de la aplicación, realizar pruebas

y probar el framework por medio de la compilación y ejecución del código fuente. A continuación se describe el entorno de trabajo de este tipo de usuario final:

- ✓ *Limitaciones de entorno*
  FV-GAIA 2.0 está desarrollada bajo Java y ZVTM, el usuario final podrá hacer uso de este framework por medio de un entorno de desarrollo destinado para este lenguaje de programación.

- ✓ *Integraciones con otros sistemas*
  Un usuario puede utilizar el framework con el fin de integrarlo a herramientas CASE en desarrollo o herramientas CASE que deseen mejorar la visualización de diagramas de software, para lo cual, el usuario abrirá la aplicación por medio de un entorno de desarrollo (IDE) que le permita editar cada una de las clases y paquetes que tenga el componente de integración que comunica FV-GAIA 2.0 con el software con el que se va a integrar.

d) Característica del Producto

Dada la definición de producto descrita con anterioridad, en el documento visión también se definieron algunas de las características más relevantes de FV-GAIA2.0, como son las siguientes:

- Estaría compuesto por dos editores visuales: Editor de Sintaxis Concretas y Editor de Diagramas. Este último no hace parte del framework solo es un entorno de pruebas para asegurar el funcionamiento y funcionalidades del primer editor.
- El editor de sintaxis concretas es donde el usuario final podrá manipular por medio de una barra de herramientas las diferentes primitivas gráficas para la creación y modificación de elementos de diagramas de software.
- Editor de Diagramas es donde el usuario podrá crear diagramas a partir de los elementos u objetos creados en el editor de sintaxis concretas.

II – Documento SRS

El documento SRS (Software Requirement System) fue el segundo entregable del trabajo de grado, el cual pertenece a la fase de concepción de AUP [9]. Este documento permitió hacer la especificación de requerimientos con el fin de tener los insumos correctos para el comienzo del desarrollo de software y cumplir con las necesidades del cliente.

a) Alcance de FV-GAIA2.0

Luego de definir la problemática y definir FV-GAIA 2.0 como producto de software se determinó el alcance de este framework en este documento. Se estableció que FV-GAIA 2.0 va a ser el producto de aplicación final de este trabajo de grado el cual comprende: implementar la persistencia de sintaxis concretas, permitir la administración de plantillas de un lenguaje de

modelado e implementar dos herramientas de software; un editor de Sintaxis Concretas y un Editor de Diagramas.

En términos de ZoomEnv, con la implementación a realizar en FV-GAIA 2.0 se cubrirían los siguientes elementos de la arquitectura de ésta herramienta: Administrador de Sintaxis Concreta, Enlace Vista y Sintaxis Concreta, Visualización ZUI, Editor de Sintaxis Concretas y Editor de Diagramas.

b) Estudio de Requerimientos Funcionales

Teniendo en cuenta el alcance definido, y que este nuevo framework partiría de su versión anterior, se inició el análisis de requerimientos estableciendo los requerimientos necesarios para cumplir con el alcance planteado y el objetivo general del proyecto. Para esta definición de requerimientos se realizó un estudio de los requerimientos ya establecidos para FV-GAIA y se definieron nuevos con ayuda y aprobación del director del trabajo de grado. Estos requerimientos se clasificaron en Funcionales y no funcionales, y por editor.

Una vez establecidos los requerimientos del aplicativo, se continuó el proceso de análisis de requerimientos con la priorización de los mismos, este paso tenía como objetivo hallar los requerimientos más importantes para implementar al iniciar el desarrollo del framework y cuales pueden dejarse para después, este procesos se realizó basado en Wiegers et al [26], el cual define el proceso de priorización basado en criterios de valor, costo y beneficio.

Continuando con el análisis de requerimientos, se realizó el grafo de dependencias, el cual permitió establecer los diferentes grupos de requerimientos como mostrar al desarrollador el posible impacto que tenga un cambio en determinado requerimiento, el camino de dependencias que se tiene que seguir para llegar a un requerimiento, y el orden que da un requerimiento según sus dependencias que tenga.

Complementando el proceso de análisis de requerimientos se estableció una documentación, una medición y una trazabilidad de cada uno de los requerimientos a lo largo del desarrollo del producto de software.

Para la documentación y medición de requerimientos, estos se documentaron en un formato específico donde se relacionaban los siguientes campos: Numero de Requerimiento, Descripción, Responsable, Criterio de medición, Prioridad, Versión, Formato de implementación, Volatilidad, Tiempo (días), Tipo de Requerimiento, Grupo Coherente, Fecha, Requerimientos necesarios, Riesgos Asociados y Estado, lo que permitió tener un control muy detallado sobre cada uno de los requerimientos establecidos.

Por último, para la trazabilidad de requerimientos, a medida del desarrollo se completó una matriz para este fin basado en [27] [28] [29], esta matriz permitió establecer cada uno de los paquetes, clases y métodos en donde se implementó cada método, y en si es un control para medir tanto la implementación del framework como las pruebas realizadas.

c) Atributos de Calidad

Una de las premisas de FV-GAIA 2.0 es cumplir con ciertos atributos de calidad para cumplir con las expectativas de un producto visual para herramientas CASE por lo cual se definieron los siguientes:

- Rendimiento
  - FV-GAIA 2.0 debe estar en capacidad de desplegar diagramas de software en cada uno de los niveles de detalle de forma fluida, es decir que no presente demoras en la presentación de datos ni que estas demoras impidan la visualización correcta de los diferentes diagramas.
  - La navegabilidad entre niveles de detalle y diagramas de software debe ser animada al navegar por medio de zoom. Como también fluida en el sentido de presentar problemas de procesamiento de información que impidan la visualización de diagramas.

- Usabilidad:
  - FV-GAIA 2.0 debe presentar interfaces de usuario que tengan una barra de herramientas y botones que guíen al usuario en su uso.
  - Las interfaces de usuario deben estar basadas en las interfaces de herramientas CASE cómo: StartUML [30], NetbeansIDE [3|], Dia [32], JDeveloper [33], y Together [34], entre otros, pero con mejoras que permita hacer uso de las funciones de la aplicación en muy corto tiempo.
  - El usuario al ver la interfaz, los iconos, botones y barras de herramientas, estos deben ser explícitos y tener textos de ayuda que le permita al usuario familiarizarse lo más rápido posible con las interfaces gráficas que hagan parte de FV-GAIA 2.0.

- Mantenibilidad
  - En cada uno de los componentes se debe asegurar el bajo acoplamiento y la alta cohesión para que en el tiempo, la aplicación se pueda seguir manteniendo y actualizando después de terminada cada una de las versiones.

d) Requerimientos de Producto

Una vez se definido el alcance del framework se pasó a describir las funcionalidades de cada uno de los editores que componían este aplicativo con ayuda de los requerimientos establecidos, estos se presentan en la Tabla 2.

Editor de Sintaxis Concretas
✓ Cargar, editar y guardar plantillas de sintaxis concretas. ✓ Creación y manipulación de primitivas gráficas. ✓ Transformaciones geométricas de primitivas gráficas. ✓ Crear grupos y contenedores de primitivas gráficas.

✓ Distintas formas de distribución de elementos por medio de diferentes tipos de layouts para grupos y contenedores de primitivas gráficas. ✓ Distintas formas de representación de elementos por medio de diferentes tipos de primitivas gráficas y rangos de visualización. ✓ Utilización del zoom semántico para manejar diferentes layouts de grupos, de contenedores y representaciones de elementos. ✓ Navegación a través de zoom y panning.
Editor de Diagramas
✓ Cargar plantillas de sintaxis concretas. ✓ Instanciar plantillas de sintaxis concretas. ✓ Crear nodos y aristas por medio de instancias de plantillas de sintaxis concretas ✓ Eliminar y modificar instancias de plantillas de sintaxis concretas. ✓ Crear grupos de elementos, y elementos contenedores. ✓ Transformaciones geométricas a nodos, aristas y grafos. ✓ Manipular el layout de elementos y diagramas. ✓ Navegación a través de zoom y panning. ✓ Distintas formas de distribución de elementos por medio de diferentes tipos de layouts de grafos. ✓ Cargar, editar y guardar diagramas de elementos.

Tabla 2: Requerimientos de Producto

III – Documento SDD

El documento SDD (Software Design Document) fue el tercer entregable del trabajo de grado, el cual pertenece a la fase de elaboración de AUP [9]. Este documento permitió establecer el diseño, la arquitectura de software y los patrones asociados, de acuerdo a los requerimientos establecidos, a la arquitectura que tenía FV-GAIA [7], ZVTM [15] y ZooMEnv [3])

a) Perspectiva Global

Como primer paso para la definición de la arquitectura de software y sus diseños asociados, se definió la arquitectura de software de FV-GAIA2.0 en una perspectiva global. Esta arquitectura, se encuentra gobernada por el estilo arquitectural de capas [35], y un patrón MVC [36] dentro de estas capas.

Esta escogencia de estilos arquitecturales se realizó dada la siguiente conclusión: FV-GAIA 2.0 pretende ser una aplicación con una interfaz gráfica sofisticada y poco almacenamiento de datos, es decir que la aplicación tendrá más cambios en la parte grafica con más frecuencia que en el almacenamiento de datos y la parte grafica estará relacionada directamente con la lógica de la aplicación, así que los cambios que se realizan con más frecuencia en la aplicación son en el componente gráfico y el componente de lógica.

Para este tipo de aplicativos con el comportamiento descrito con anterioridad, lo usual es utilizar el patrón Modelo-Vista-Controlador, pero como FV-GAIA 2.0 va a tener varios componentes adicionales, el estilo arquitectural a utilizar será por capas y dentro de estas si se utilizará MVC. Dada la anterior explicación en la Ilustración 8 se presenta la arquitectura de software que caracteriza a FV-GAIA 2.0, y a continuación se explican los componentes que hacen parte de esta.

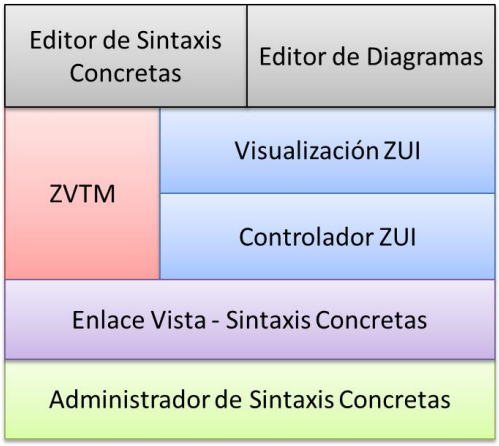


Ilustración 8: Arquitectura Global de FV-GAIA2.0

- *Editor de Sintaxis Concretas:* Permitirá la creación y manipulación de meta-modelos de elementos de diagramas de software a partir de primitivas gráficas.
- *Editor de Diagramas:* Permitirá la creación y manipulación de modelos de software a partir de meta-modelos definidos en el editor para este fin.
- *Visualización ZUI:* Presenta toda la funcionalidad que permite visualizar diagramas de software y navegar a través de ellos por medio de zoom y panning.
- *Controlador ZUI:* Presenta toda la funcionalidad que permite la creación, manipulación y modificación de elementos de los modelos de software que se pueden visualizar en el plano de trabajo.
- *ZVTM:* Es el framework elegido como candidato para el desarrollo de FV-GAIA [7]. Presenta el manejo de primitivas gráficas, manejo de cámaras en 2D y la implementación de interfaces de usuario aumentables.
- *Enlace Vista - Sintaxis Concretas:* Es el encargado de mantener actualizadas y sincronizadas los objetos e instancias que almacena las sintaxis concretas con las clases que representan los diagramas de software visualmente en la interfaz gráfica e interfaz de usuario aumentable.
- *Administrador de Sintaxis Concretas:* Es el encargado de almacenar e instanciar las sintaxis concretas creadas en el editor, también es el responsable de asociar las sintaxis concretas a los elementos de los modelos de software a crear en el editor de diagramas.

b) Diagrama de Componentes

De acuerdo a la arquitectura presentada de forma global en la sección anterior, el siguiente paso fue la realización del diagrama de componentes y los diagramas de paquetes y clases. El diagrama de componentes se presenta en la Ilustración 9, este muestra en detalle los componentes y las relaciones que hacen parte de la arquitectura de FV-GAIA 2.0, estos componentes se describieron en la sección anterior.

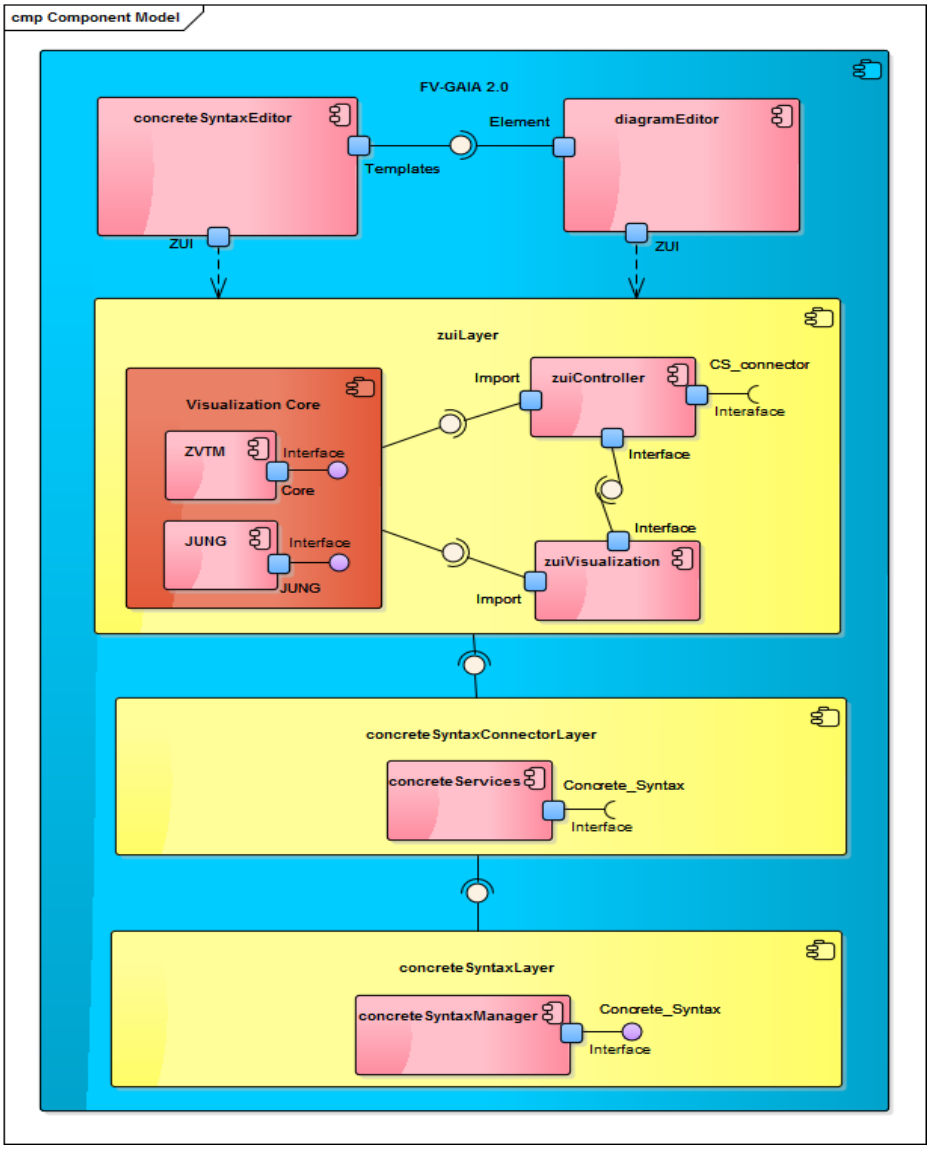
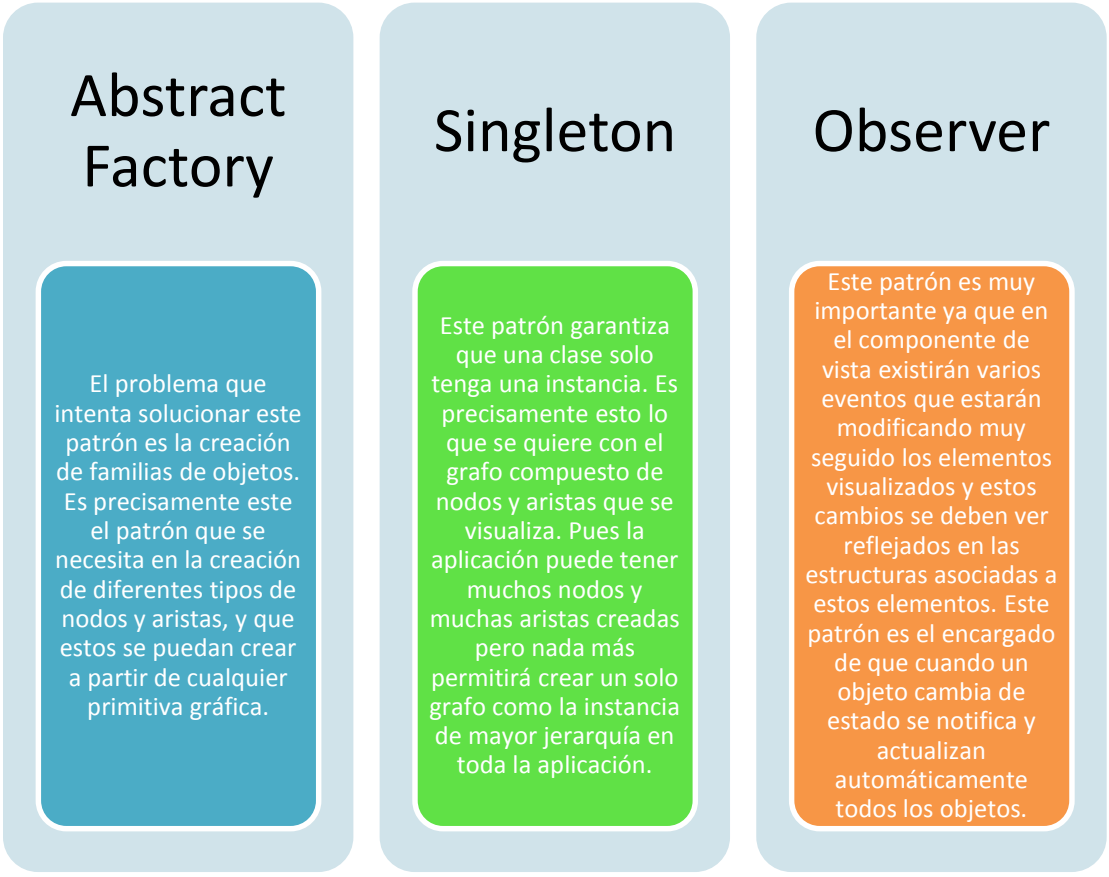


Ilustración 9: Diagrama de Dominio FV-GAIA 2.0

c) Diseño de Bajo Nivel

Una vez especificado el diagrama de dominio y el diagrama de componentes se realizó el diagrama de clases y paquetes pero dado el tamaño de software y de los componentes este se dividió por cada una de las capas que presenta el framework. Para cada una de estas capas se realizó un diagrama de clases y paquetes junto con la descripción de cada una de las clases que lo componían. Para esta etapa del diseño y para el diseño en general de FV-GAIA 2.0 se reutilizó la mayoría de las clases y paquetes de FV-GAIA [7], adaptándolas a las necesidades de los nuevos requerimientos, como también se generaron varias clases nuevas para cada una de las capas que no existían en FV-GAIA [7].

Para las capas de los editores, la capa ZUI y el conector de sintaxis concretas se utilizaron varios patrones de diseño para lograr el comportamiento que se deseaba en ciertos objetos de acuerdo a los requerimientos establecidos para el framework. Los patrones a utilizar se describen en la Tabla 2.



Para la capa de sintaxis concretas, el diseño de esta capa se basó en los modelos y diagramas provistos por el documento Diagram Definition de OMG [8]. Como se explicó en el marco teórico, Diagram Definition es una especie de framework que por medio de modelos base permite definir diagramas para la construcción de las estructuras y objetos que permitirán

construir la sintaxis concreta de un elemento de diagrama, y esto mismo es lo que se requería para esta capa, pero con un componente adicional, y era no solo la utilización de estos modelos para la representación de sintaxis concretas, sino extender estos modelos para representar sintaxis concretas en interfaces aumentables. Los modelos y clases de Diagram Definition se muestran en la Ilustración 10.

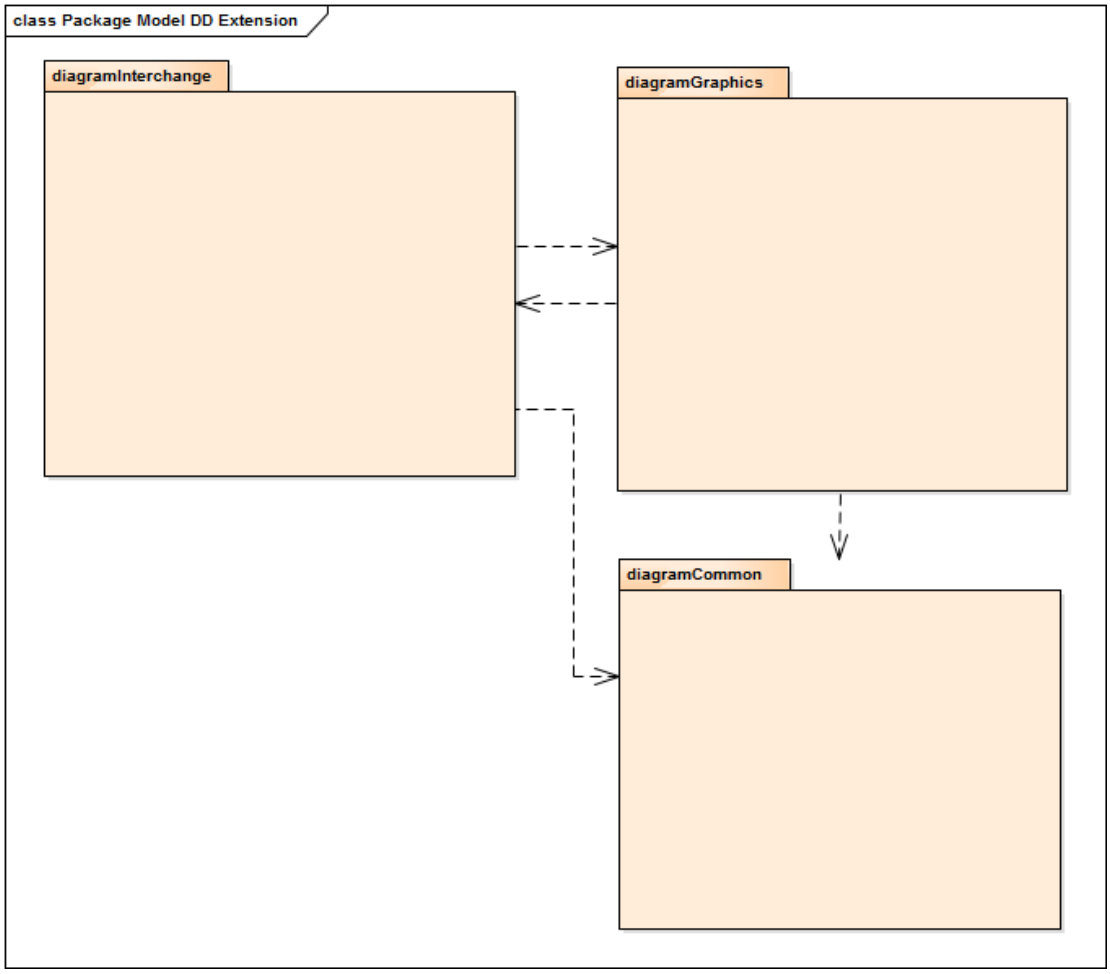


Ilustración 10: Diagram Definition

Basado en la premisa anterior, se realizó el diseño de esta capa tomando como base en Diagram Definition [8], pero para su utilización dentro de FV-GAIA 2.0 se realizaron varios ajustes a los modelos originales generando así, no solo atributos y clases adicionales sino también varios componentes adicionales que permitían la comunicación y administración de los nuevos modelos para la representación de sintaxis concretas que se generaban en la capa visual de FV-GAIA 2.0.

Como resultado de las modificaciones y generaciones de nuevos componentes de software a los diagramas originales de Diagram Definition [8], en la Ilustración 11 se muestra el diagra-

ma de paquetes resultante. Este a su vez es uno de los productos finales del presente trabajo de grado, el cual está alineado con el objetivo específico 2.

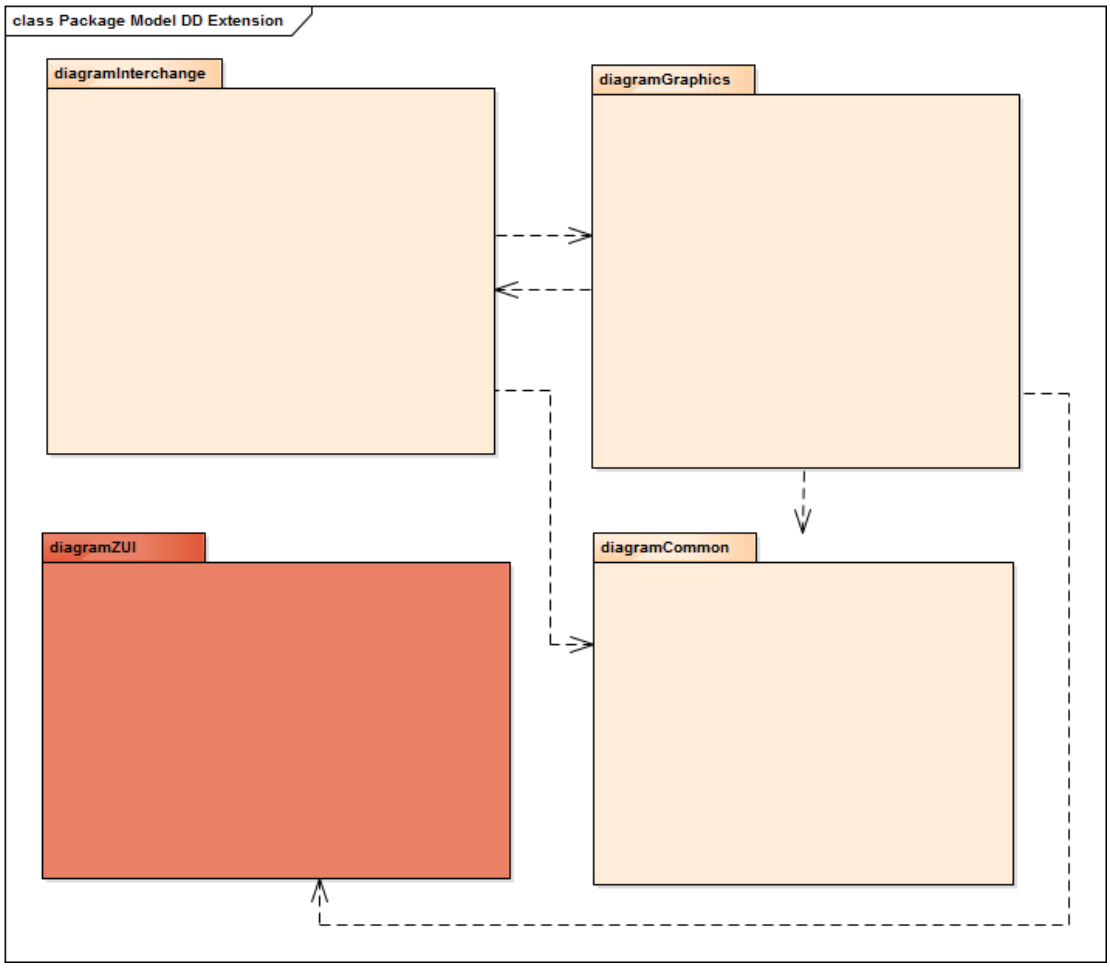


Ilustración 11: Diagram Definition extendido

El diagrama anterior muestra solo los paquetes que tiene la versión original de Diagram Definition más un nuevo paquete. En las siguientes ilustraciones se puede apreciar cada uno de los cambios realizados a las clases y atributos existentes, como las nuevas clases y paquetes agregados para la representación de sintaxis concretas aplicadas a interfaces aumentables.

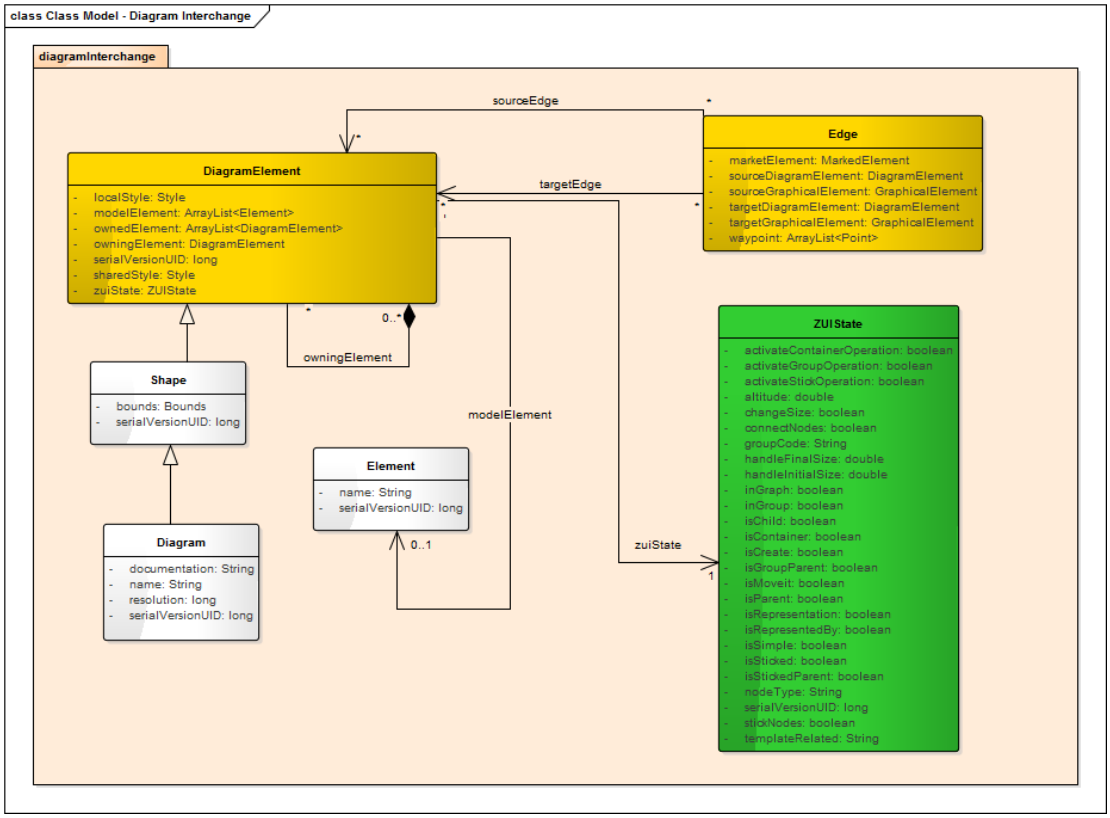


Ilustración 12: Diagrama de clase y paquetes DiagramInterchange

En la Ilustración 12 se muestra el diagrama de clases y paquetes de Diagrama Interchange, un paquete original de Diagrama Definition, el cual se encuentra modificado. Las clases que se encuentran en amarillo son clases a las cuales se les agregaron atributos de Diagram Definition y ZUI, y la clase que se encuentra en verde es una clase nueva que representa el estado de una sintaxis concreta.

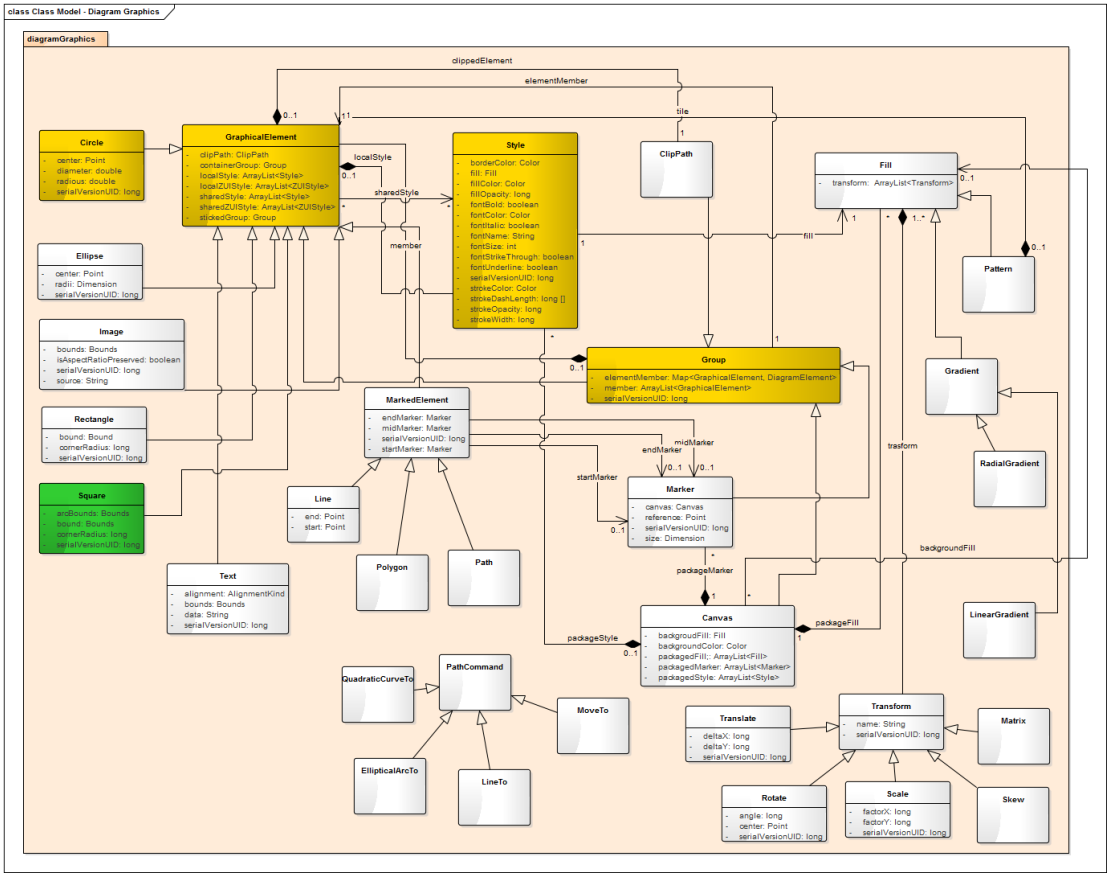


Ilustración 13: Diagrama de clase y paquetes Diagram Graphics

En la Ilustración 13 se muestra el diagrama de clases y paquetes de Diagrama Graphics, un paquete original de Diagrama Definition, el cual se encuentra modificado. Las clases que se encuentran en amarillo son clases a las cuales se les agregaron atributos de Diagram Definition y ZUI, y la clase que se encuentra en verde es una clase nueva que representa las especificaciones de una figura gráfica en forma de cuadrado.

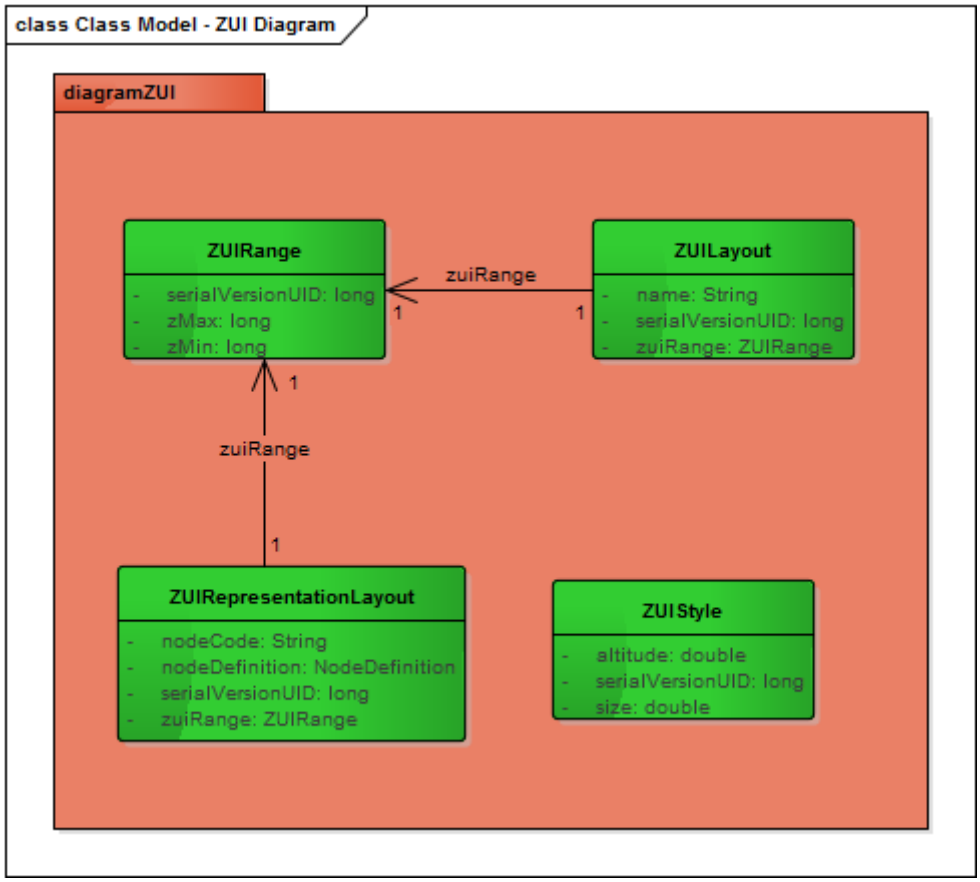


Ilustración 14: Diagrama de clases y paquetes Diagram ZUI

Este paquete es nuevo en Diagrama Definition, se relaciona con Diagram Graphics, y se llama diagramZUI. Contiene clases encargadas de: administrar layouts, zoom semántico, rangos de representación, administración de estilos y administración del comportamiento o estado de las diferentes sintaxis concretas.

Como complemento a lo descrito anteriormente, a continuación se hace un listado de cada una de las clases de Diagram Definition [73] y su extensión que fueron utilizadas para la generación y conversiones de sintaxis concretas en FV-GAIA 2.0

- Diagram Common: AlignmentKind, Bounds, Color, Dimension, y Point.
- Diagram Interchange: DiagramElement y Edge.
- Diagram Graphics: Circle, ClipPath, Ellipse, Fill, GraphicalElement, Group, Image, Line, MarkedElement, Rectangle, Square, Style, Text
- Diagram ZUI: ZUILayout, ZUIRange, ZUIRepresentationLayou, ZUIState, ZUIStyle,

IV – Productos de Software FV-GAIA 2.0

Cada uno de los entregables descritos con anterioridad se realizaron con el propósito de especificar lo mejor posible, en términos de análisis y diseño los desarrollos de software producto de este trabajado de grado. En este apartado se describirán los productos de software generados como sus principales funcionalidades y su propósito.

a) Editor de Sintaxis Concretas

I. Propósito y uso

Este Editor le permitirá al usuario crear plantillas de sintaxis concretas a partir de ciertas primitivas gráficas disponibles en una barra de herramientas. El usuario creará este tipo de plantillas seleccionando primitivas gráficas de esta barra de herramientas a un plano de trabajo, con el propósito de crear elementos de diagramas de acuerdo a las necesidades del usuario. En este editor, las primitivas gráficas se podrán mover, girar, escalar, editar, agrupar, contener dentro de otras, acomodarlas de forma arbitraria, agregar distintas presentaciones y layouts para luego ser guardadas como plantillas de elementos de diagramas.

II. Descripción

El editor de sintaxis concretas es el principal producto de software de este trabajo de grado. Este editor se construyó bajo el lenguaje Java, ayudado de ZVTM [15], de la primera versión de FV-GAIA [7] y teniendo en cuenta cada una de las especificaciones de diseño descritas en los entregables presentados con anterioridad.

Para el editor de sintaxis concretas se definió la interfaz gráfica de usuario que se presenta en la Ilustración 15, esta se compone de varias barras de herramientas, barras de menús, botones y un lienzo de trabajo o canvas que para que el usuario se comunique con la interfaz gráfica.

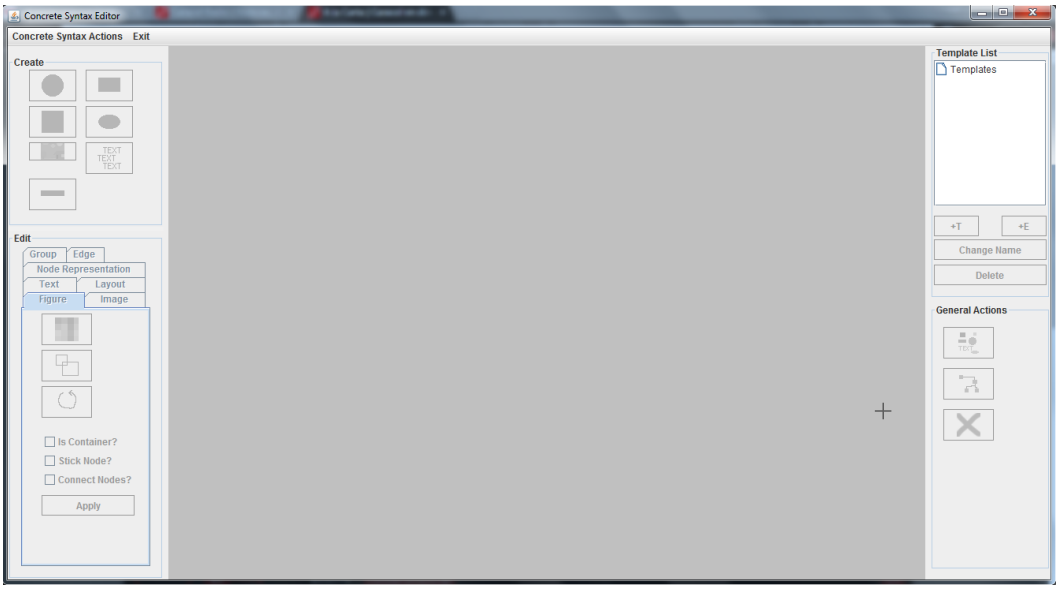


Ilustración 15: Editor de Sintaxis Concretas

III. Funcionalidades

A continuación se presenta y describen las principales funcionalidades del editor de sintaxis concretas.

- Administración de plantillas de sintaxis concretas.
  Uno de los aspectos más interesantes de este editor es el componente y concepto de plantillas de sintaxis concretas que utiliza. Este se basa en la posibilidad de crear varios grupos de documentos de plantillas dentro del editor, los cuales se alojan de forma temporal en memoria pero a su vez se guardan de forma persistente, una vez que lo indique el usuario. Cada uno de estos documentos aloja cada una de las primitivas gráficas y plantillas de sintaxis concretas que el usuario cree. Este sistema de plantillas funciona de la misma forma que el sistema de archivos de Windows, en carpetas y archivos simples.

  Este sistema de plantillas de sintaxis concretas se gestiona desde la interfaz gráfica de usuario en la parte derecha superior como se muestra en la Ilustración 16. En este panel se administra la creación, edición y eliminación de plantillas y documentos de sintaxis concretas. Estos documentos son la base para la creación de primitivas gráficas y configuración de sintaxis concretas ya que sin estos, las acciones de creación no se habilitarán.

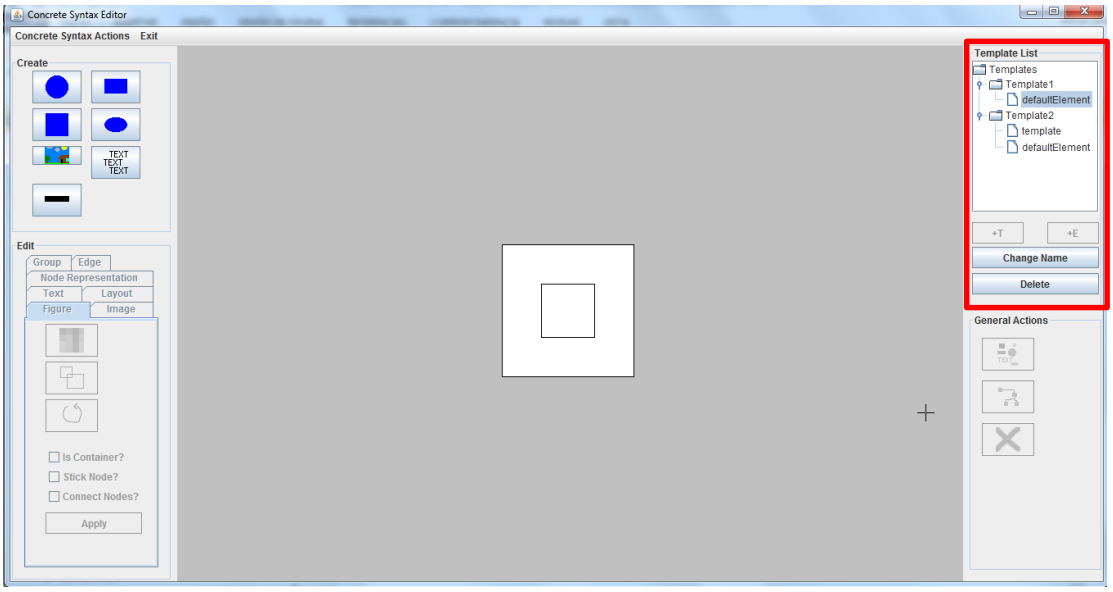


Ilustración 16: Administración de plantillas de sintaxis concretas.

- Creación y manipulación primitivas gráficas.
  Cada uno de los elementos gráficos está basado a partir de primitivas gráficas que son los que se crean en este editor, estos se pueden apreciar en la Ilustración 17. Estas primitivas gráficas se crean por medio del panel de creación (resaltado en rojo) de la interfaz gráfica, haciendo clic en cualquiera de esos botones, estas se manipulan por medio del mouse, y se edita su contenido o apariencia por medio del panel de edición (resaltado en verde) de la interfaz gráfica. Cada una de estas primitivas gráficas se guarda en un documento de plantilla de sintaxis concretas.

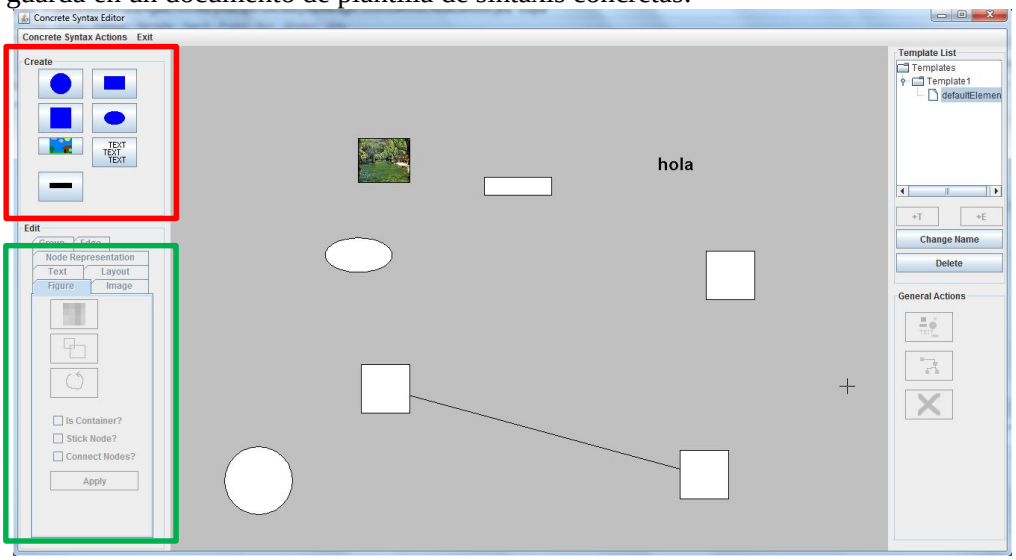


Ilustración 17: Creación y manipulación de primitivas gráficas

- Creación de grupos y contenedores de primitivas gráficas.
  A partir de varias primitivas gráficas se pueden crear elementos gráficos más complejos como contenedores de primitivas gráficas o grupos de primitivas. Estos en si actúan como cualquier otro elemento pero tiene características especiales; los contenedores pueden contener otros elementos en su interior y los grupos, agrupan varias primitivas gráficas. En la Ilustración 18 se puede observar cómo se comportan los contenedores (resaltado en verde) y grupos (resaltado en rojo) de primitivas gráficas.

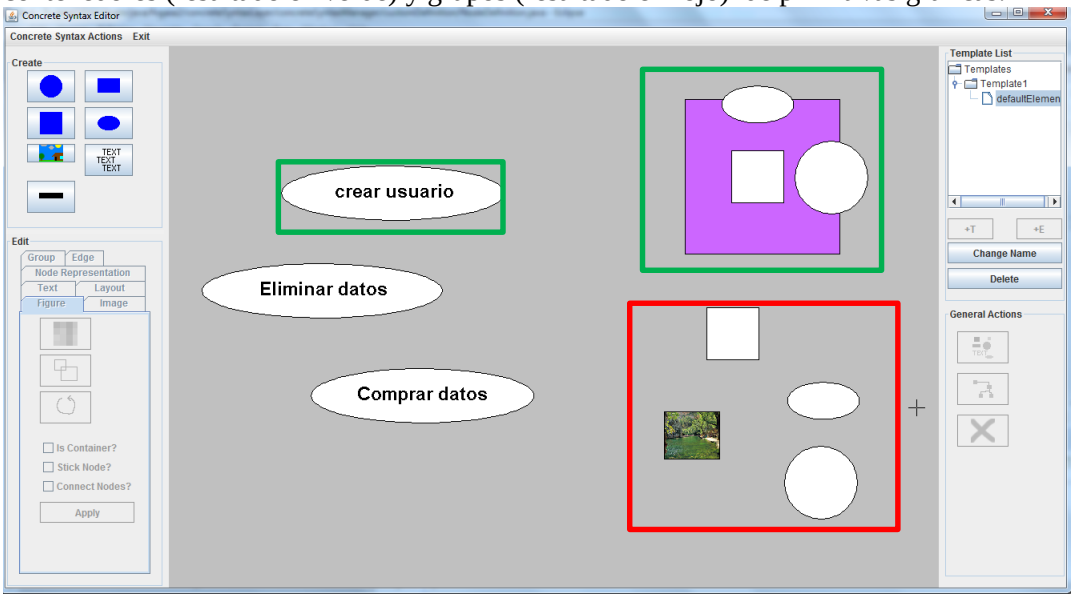


Ilustración 18: Grupos y contenedores de primitivas gráficas

- Aplicar distintos layouts de elementos.
  Para los grupos y contenedores de primitivas gráficas presentados con anterioridad es posible cambiar la organización de sus elementos aplicando o agregando diferentes layout de grafos por medio del panel de edición de la interfaz gráfica para que estos actúen inmediatamente sobre los elementos o actúen solo en cierto rango de zoom dado, utilizando el concepto de zoom semántico. Las Ilustración 19 muestra cómo se aplica el layout a un contenedor de primitivas gráficas. Para el ejemplo, al contenedor rectangular se le aplica un layout matricial, lo que genera que los elementos contenidos dentro de este se organicen de esta forma.

  La Ilustración 20 muestra cómo serían los layouts de grafos de acuerdo al zoom semántico. Para el ejemplo, se muestra un grupo de elementos que presenta dos layout distintos. En la imagen de la izquierda, es cuando hay mayor zoom y los elementos se organizan en forma circular, y en la imagen de la derecha, es cuando hay menor zoom y la cámara se encuentra lejos de los elementos, y estos se ordenan de forma matricial.

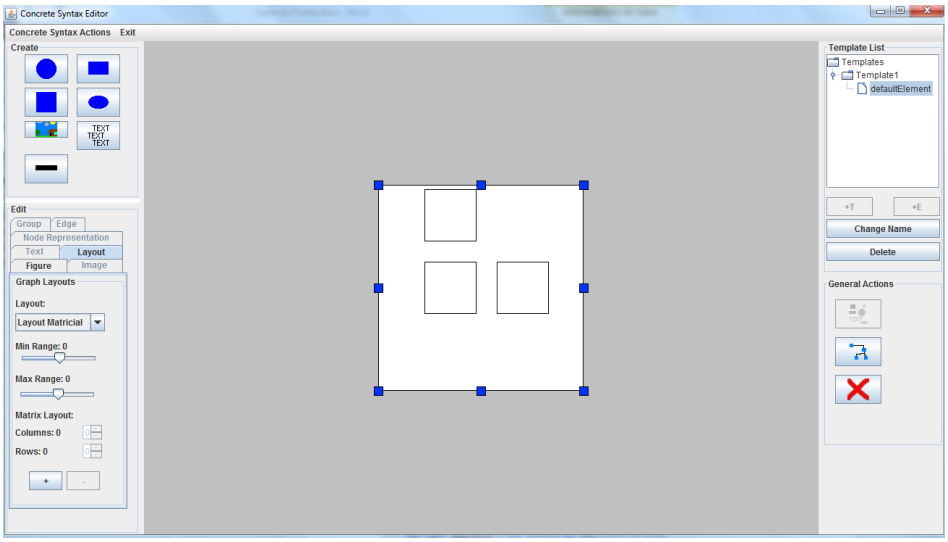


Ilustración 19: Layout aplicado a contenedores

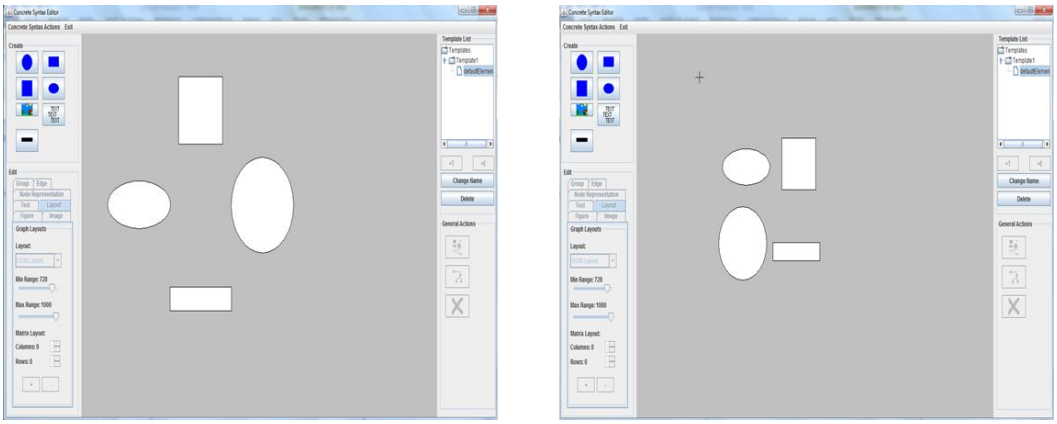


Ilustración 20: Layout aplicado en zoom semántico

- Crear distintas representaciones de elementos.
  Además de los layout aplicados a grupos y contenedores de primitivas gráficas, en este editor también es posible agregar distintas representaciones a primitivas gráficas, primitivas agrupadas o contenedores de primitivas. Con el propósito de que estas primitivas a cierto nivel de zoom o detalle tengan una representación diferente. En esta funcionalidad también se aplica el zoom semántico. En la Ilustración 21 y 22 se ve como un elemento o varios pueden tener diferentes representaciones de acuerdo al nivel de zoom. Para el ejemplo las imágenes de las izquierda representan al elemento que se muestra en las imágenes de la derecha, a un nivel bajo de zoom. A un nivel alto de zoom se muestra la representación del elemento de las imágenes de la derecha.

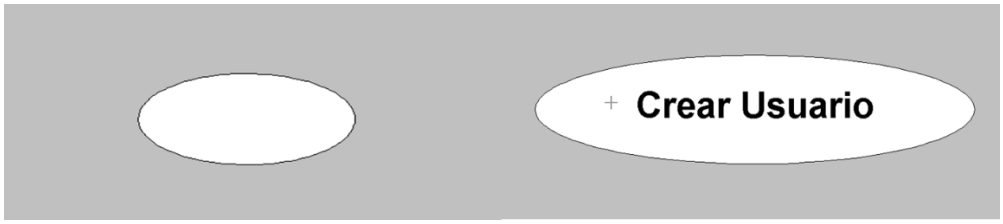


Ilustración 21: Representación de elementos

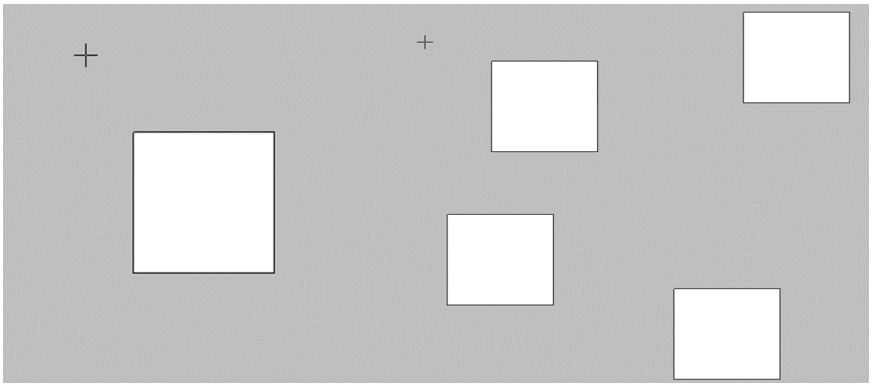


Ilustración 22: Representación de grupos

- Guardar plantillas de elementos de diagramas.
  Este editor de sintaxis concretas tiene la posibilidad de abrir y guardar en archivos planos toda la información de las plantillas sintaxis concretas, documentos y elementos visuales que se encuentra creados en el editor. Estos archivos también son los utilizados en el prototipo de pruebas para cargar las plantillas generadas en este editor.

b) Editor de Diagramas

I. Propósito y uso

Este Editor le permitirá al usuario crear elementos de diagramas y diagramas a partir de las plantillas de Sintaxis Concretas ya creadas con anterioridad. El usuario creará estos elementos de diagramas arrastrando las plantillas a un plano de trabajo donde se podrán mover, girar, escalar y acomodar de forma arbitraria para formar diagramas de software.

II. Descripción

El editor de diagramas *no es un producto de software de este trabajo de grado*, tan solo es un prueba de concepto que permitió probar cada una de las plantillas de sintaxis concretas generadas en el editor descrito con anterioridad. Este editor se construyó también bajo el lenguaje Java, ayudado de ZVTM [15], de la primera versión de FV-GAIA [7] y teniendo en cuenta cada una de las especificaciones de diseño descritas en los entregables presentados con anterioridad.

Para el editor de diagramas se definió la interfaz gráfica de usuario que se presenta en la Ilustración 23, esta se compone de varias barras de herramientas, barras de menús, botones y un lienzo de trabajo o canvas para que el usuario se comunique con la interfaz gráfica.

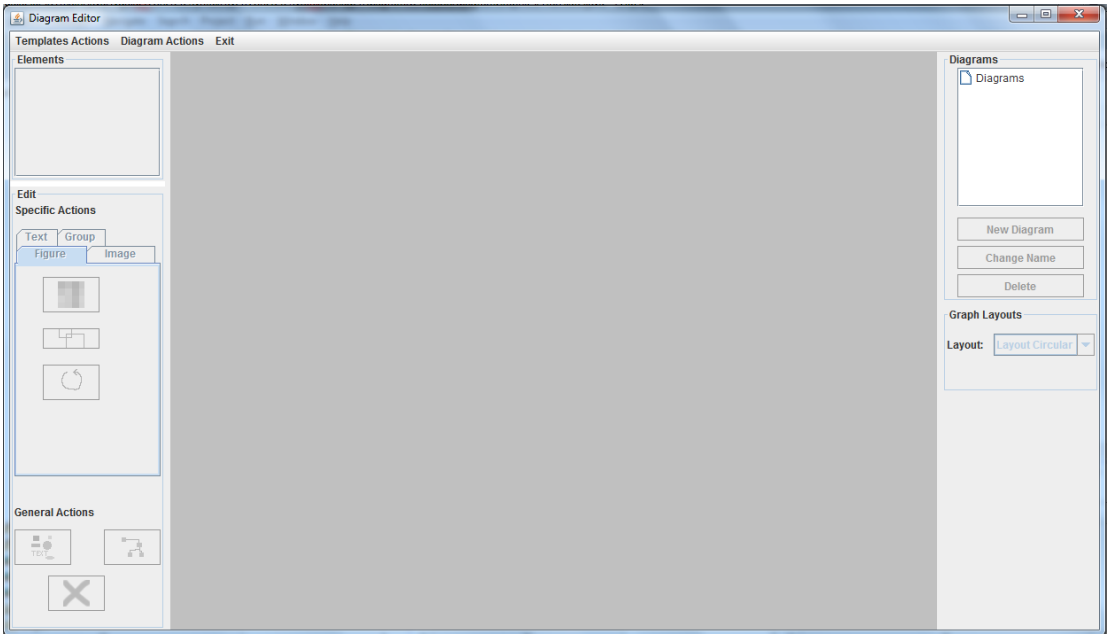


Ilustración 23: Editor de Diagramas

III. Funcionalidades

A continuación se presenta y describen las principales funcionalidades del editor de diagramas.

- Cargar plantillas de sintaxis concretas.
  Este editor de diagrama tiene la posibilidad de carga los archivos planos generados en el editor de sintaxis concretas para cargar toda la información de las plantillas sintaxis concretas, documentos y elementos visuales que se hayan creado en el editor de sintaxis concretas.

- Administración de diagramas de elementos.
  Así como el editor de sintaxis tiene su administrador de plantillas, este editor de diagramas también cuenta con un administrador de diagramas de elementos, este se basa en la posibilidad de crear varios grupos de documentos de diagramas dentro del editor, los cuales se alojan de forma temporal en memoria pero a su vez se guardan de forma persistente, de acuerdo a lo que indique el usuario. Cada uno de estos documentos aloja instancias de plantillas de sintaxis concretas que el usuario crea. Este sistema de plantillas funciona de la misma forma que el sistema de archivos de Windows, en carpetas y archivos simples.

Se diferencia del administrador de sintaxis concretas, en que en este se crean los distintos grupos de diagramas a partir de los archivos planos de plantillas de sintaxis concretas cargados. Este sistema de plantillas de diagramas se gestiona desde la interfaz gráfica de usuario en la parte derecha superior (resaltado en verde) como se muestra en la Ilustración 24. En este panel se administra la creación, edición y eliminación de documentos de diagramas. Estos documentos son la base para la creación de instancias de sintaxis concretas ya que sin estos, no se habilitarán las diferentes plantillas a crear.

En la Ilustración 24 también de muestra el ítem del menú de acciones donde se cargan los diferentes archivos de plantillas de sintaxis concretas (resaltado en rojo), y el panel de instanciación de sintaxis concretas que contiene una plantilla cargada (resaltado en amarillo). Este panel se llama Elements y se encuentra en la parte superior izquierda de la interfaz gráfica de usuario. Cada uno de los elementos que se presentan acá hace referencia a cada una de las plantillas de sintaxis concretas diseñadas y configuradas en el editor anteriormente descrito.

De acuerdo a lo anterior, los archivos de plantillas de sintaxis concretas funcionan de la siguiente manera en el editor de diagramas. Estos al ser cargados, se crean carpetas de diagramas de acuerdo al grupo de plantillas de sintaxis concretas que contenga el archivo cargado. A su vez, cada uno de estos grupos de plantillas sintaxis concretas contiene uno o varios elementos que pueden ser instanciado únicamente en la carpeta del grupo de plantillas relacionado. La Ilustración 25 muestra un ejemplo de una instanciación de elementos de sintaxis concretas simples.

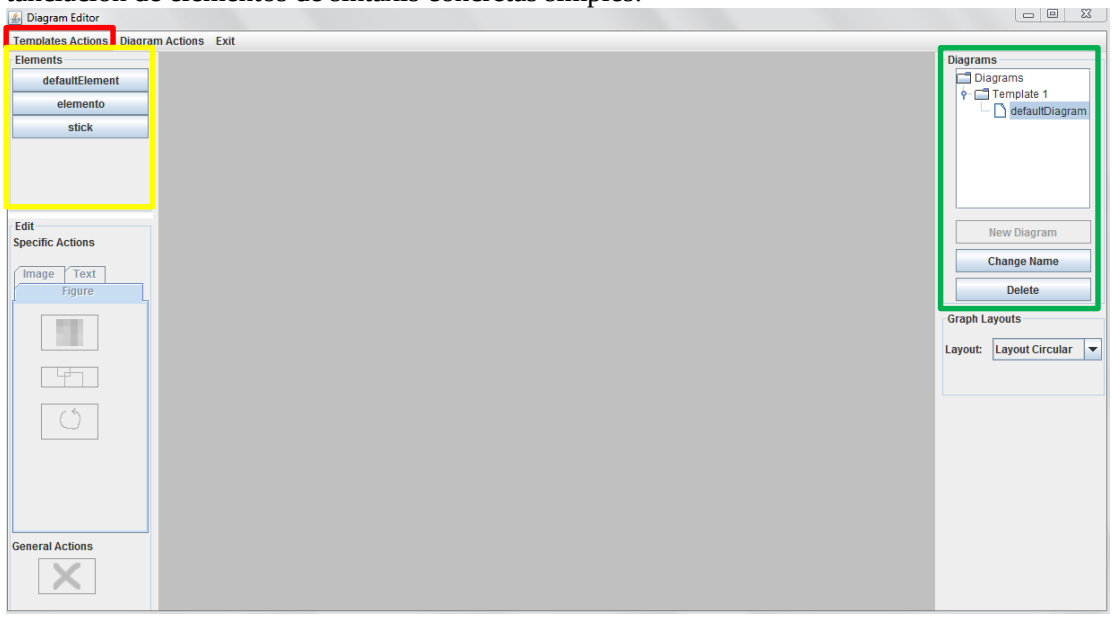


Ilustración 24: Administración de plantillas de diagramas

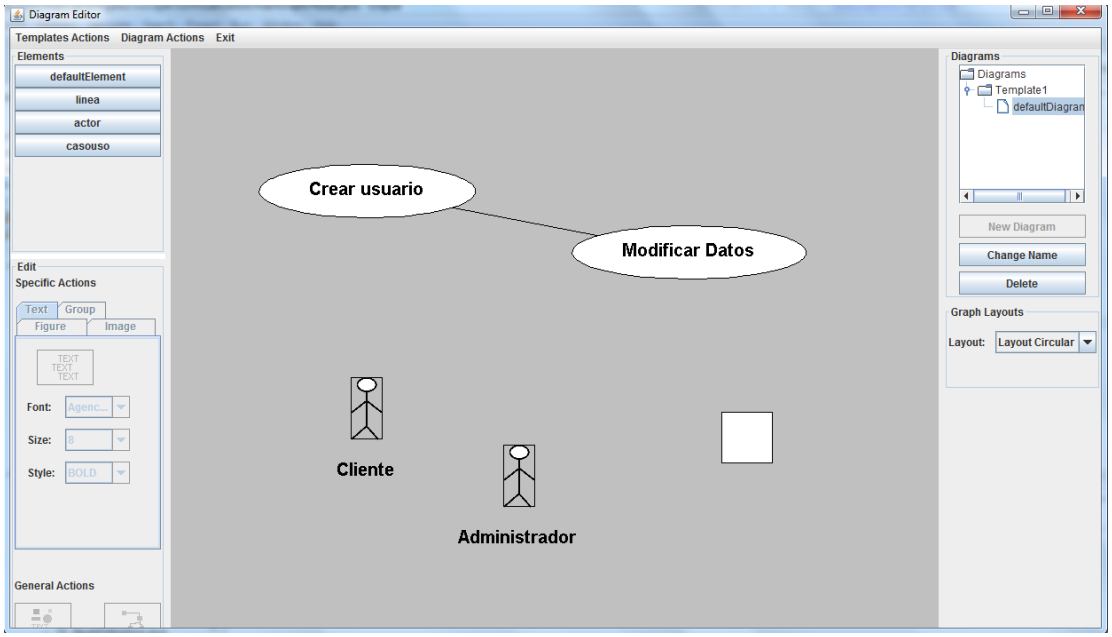


Ilustración 25: Administrador de diagramas

- Crear nodos y aristas por medio de plantillas de sintaxis concretas.
  De acuerdo a la anterior explicación, en un documento de diagrama se pueden crear nodos, y conectar nodos por medio de aristas. Estos nodos y aristas se crean a partir de instanciar plantillas de sintaxis concretas. En la Ilustración 26 y 27 se presenta como se crean nodos y aristas a partir de un archivo de plantillas de sintaxis concretas cargado, con el propósito de crear diagramas.

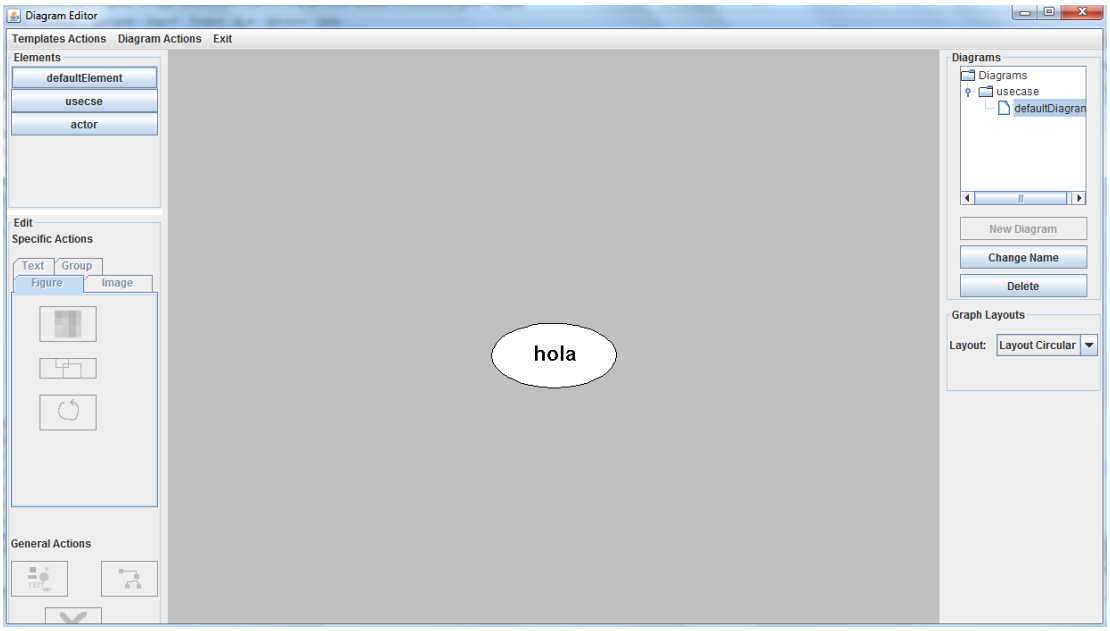


Ilustración 26: Instancia de nodo de sintaxis concretas.

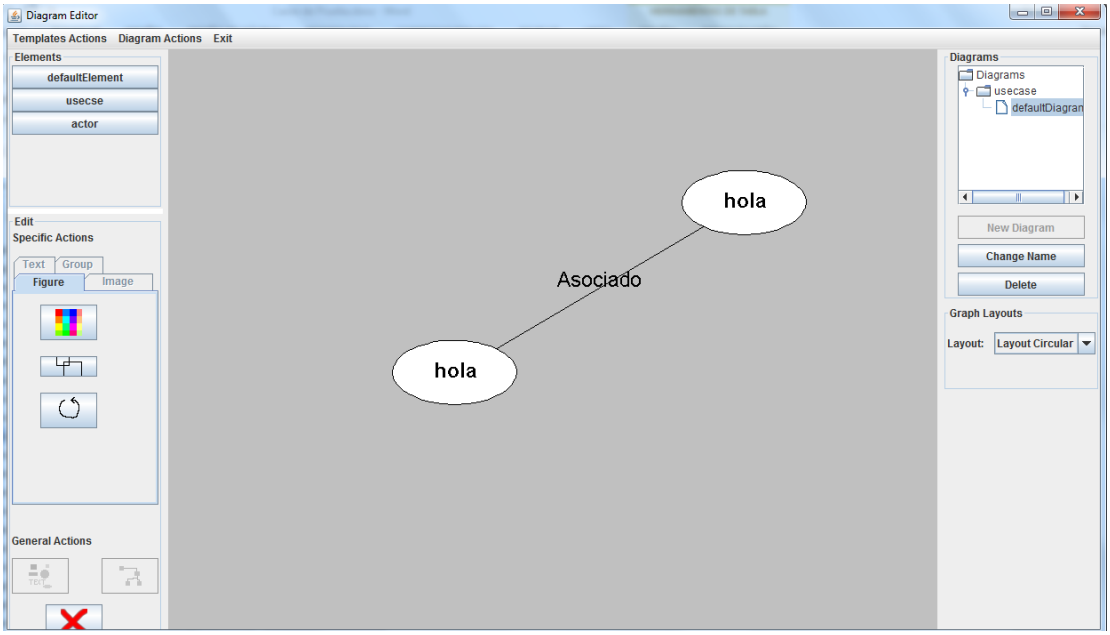


Ilustración 27: Instancia de aristas de sintaxis concretas.

- Instanciar grupos y contenedores de plantillas de sintaxis concretas.
  Este editor permite también instanciar grupos y contenedores de plantillas concretas como se muestra en las siguientes ilustraciones. La Ilustración 28 muestra un elemento de diagrama conocido, el cual es un caso de uso, pues este caso de uso se creó y configuró en el editor de sintaxis concretas a partir de primitivas gráficas, y se instan-

ció en este editor. Este caso de uso es un contenedor, donde la elipse es el contenedor y el texto es el elemento contendido, que a la vez actúa como un nodo cualquiera.

La Ilustración 29 muestra otro elemento de diagrama conocido, el cual es un actor de casos de uso, pues este caso de uso se creó y configuró en el editor de sintaxis concretas a partir de primitivas gráficas, y se instanció en este editor. Este actor es un grupo de primitivas gráficas compuesto por una imagen y un texto que permanecen unidos y en la misma organización por la configuración de grupos realizada en el editor de sintaxis concretas.

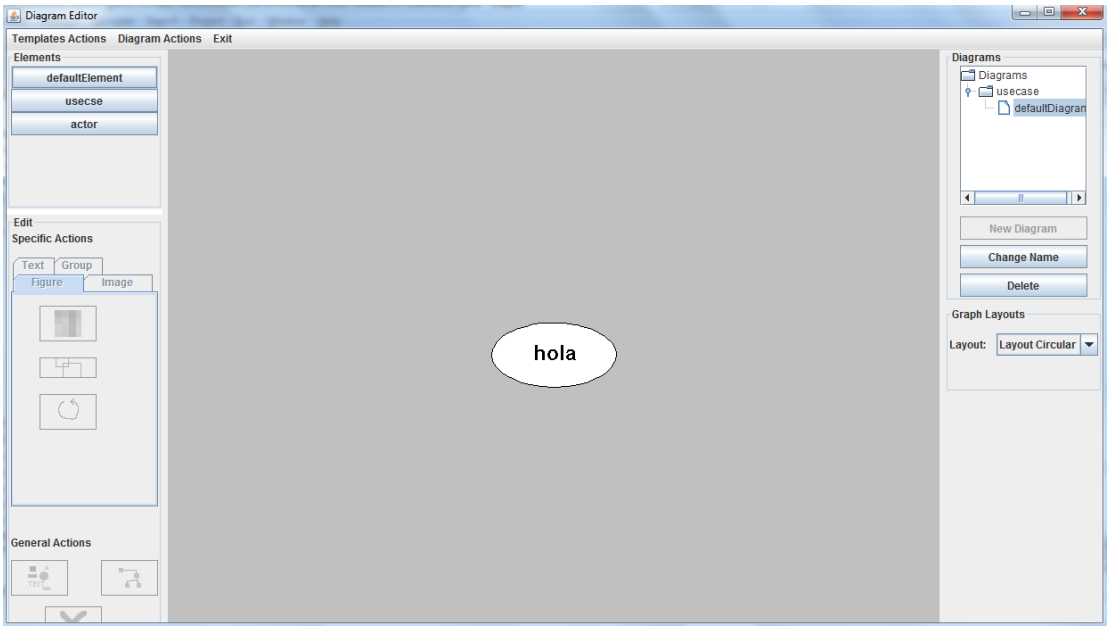


Ilustración 28: Instanciación de un contenedor

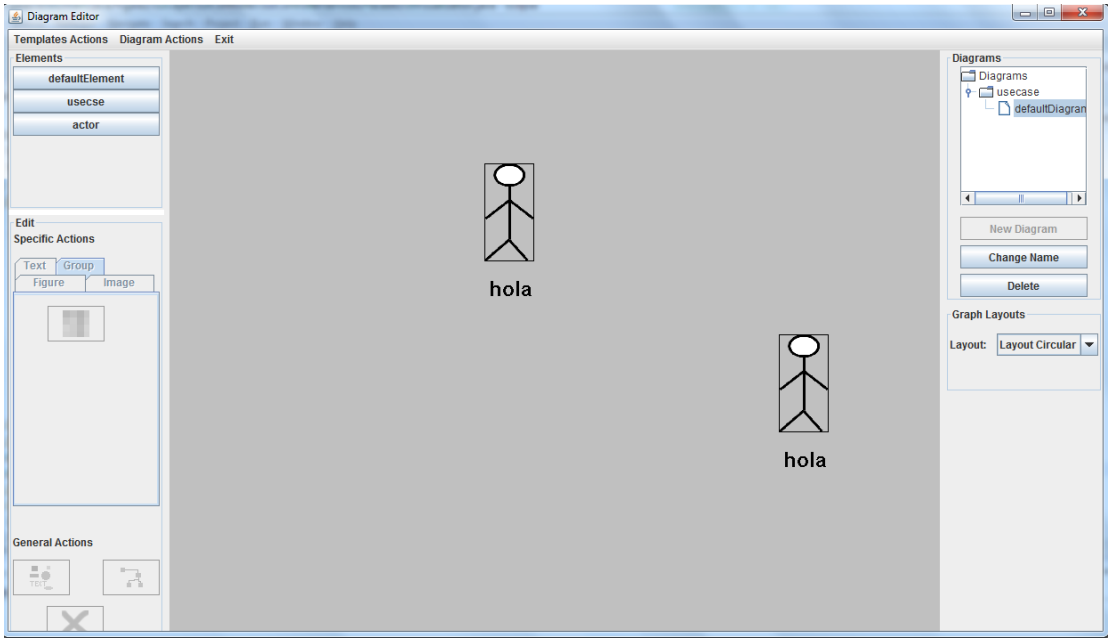


Ilustración 29: Instanciación de un grupo de nodos.

- Modificar elementos de diagramas.
  Al igual que en el editor de sintaxis concretas, en este editor también es posible editar el contenido o apariencia por medio del panel de un panel de edición. En la Ilustración 30 se presenta la ubicación del panel de edición en la interfaz gráfica de usuario de este editor (resaltada en rojo), y en la Ilustración 31 se muestra algunas instancias de plantillas modificadas.

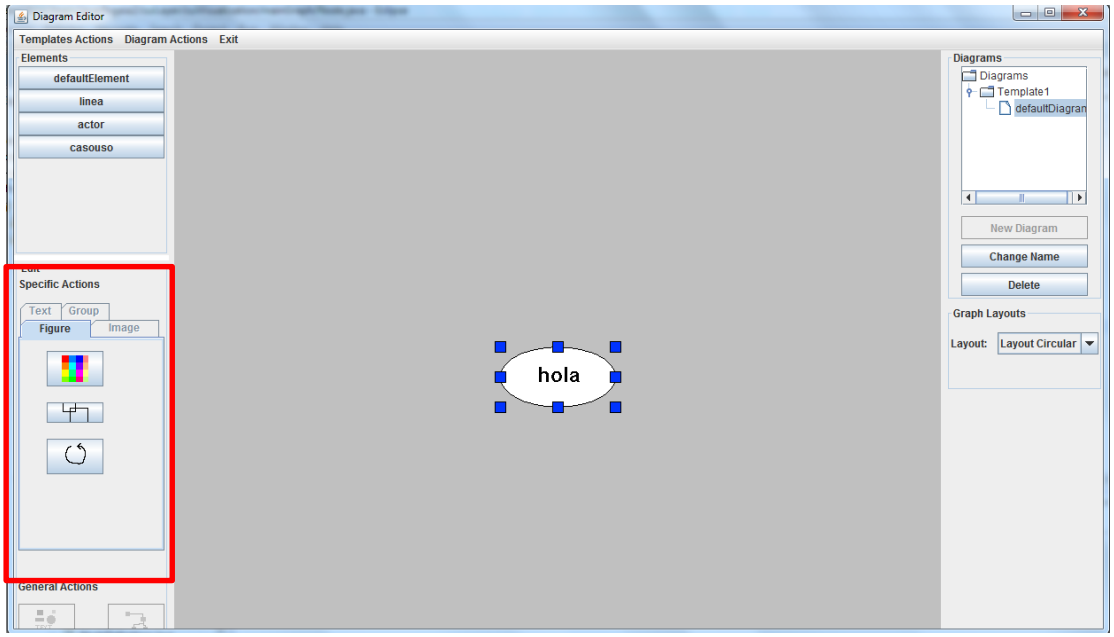


Ilustración 30: Panel de edición de plantillas de sintaxis concretas

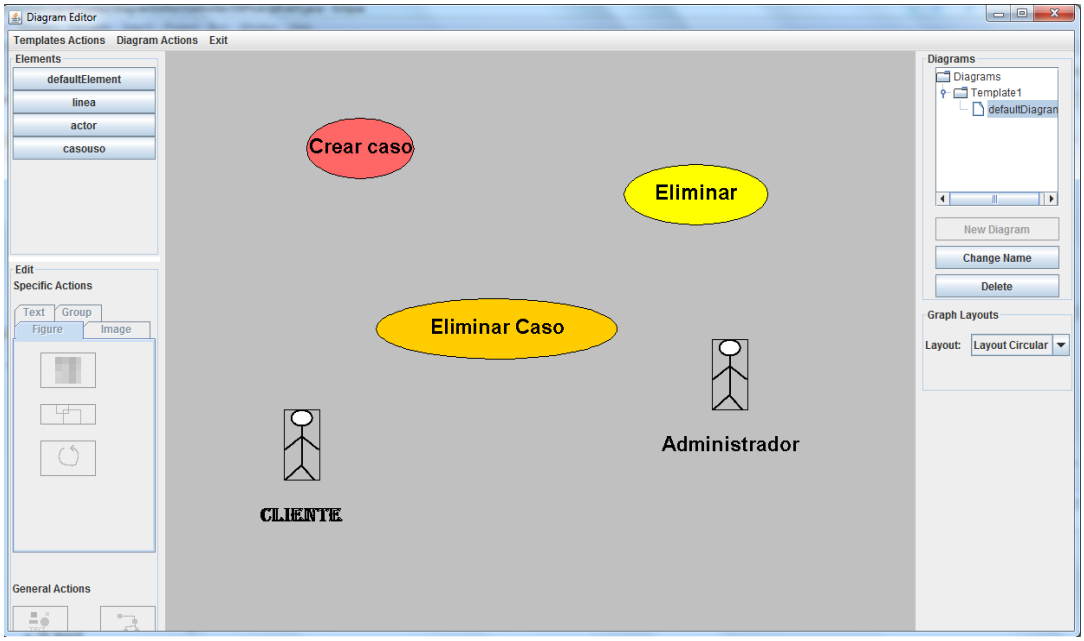


Ilustración 31: Edición de plantillas de sintaxis concretas

- Crear grupos de elementos, y elementos contenedores.
  Al igual que en el editor de sintaxis concretas, en este editor también es posible agrupar varios elementos de instancias de plantillas. En la Ilustración 32 se presentan dos imágenes, en la de la izquierda de muestran los nodos seleccionados a agrupar y en la

imagen de la derecha se muestra el resultado de agrupación de los nodos seleccionados.

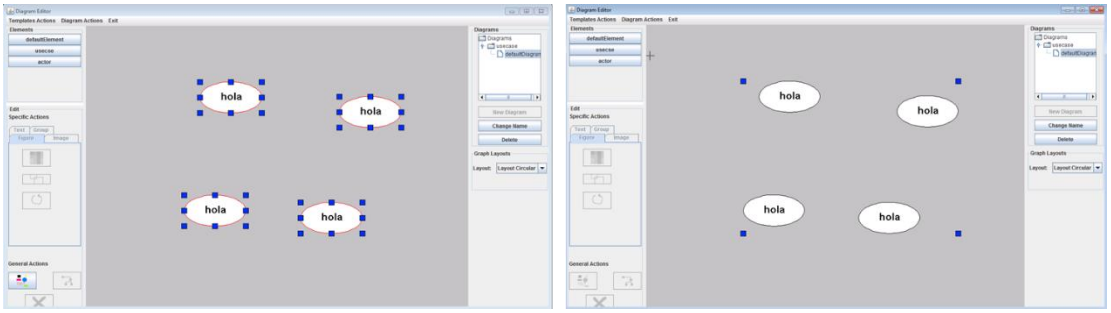


Ilustración 32: Grupos de plantillas de sintaxis concretas.

En el editor de sintaxis concretas también es posible configura el comportamiento de los elementos para que estos puedan ser contenedores de nodos y permitan conexiones. En la Ilustración 33 se presenta un contenedor de plantillas de nodos, y en la Ilustración 34 se presentan dos nodos que permiten conexiones con otros nodos.

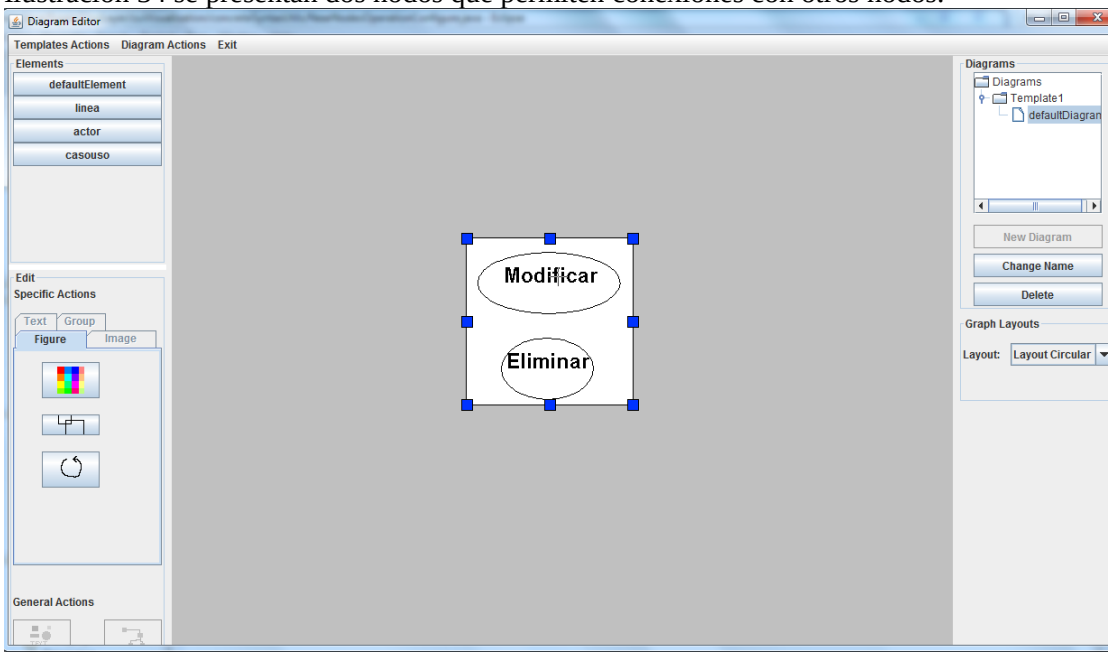


Ilustración 33: Contenedor de plantillas de nodos

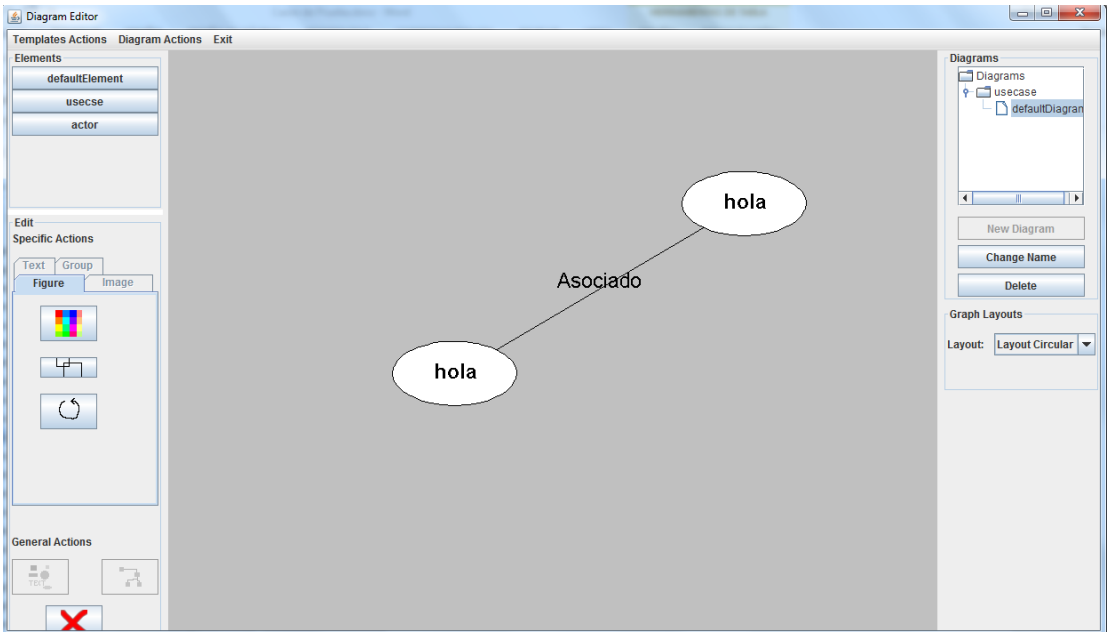


Ilustración 34: Nodos que permiten conexiones.

- Aplicar layout de elementos
  Así mismo como en el editor de sintaxis concretas se configuran layout para que se aplique a los elementos de los grupos de nodos o contenedores de acuerdo a la altura de la cámara, en el editor de diagramas al instanciar este tipo de elementos se puede comprobar que tiene diferentes layout haciendo zoom in o zoom out con el botón de scroll del mouse. La Ilustración 33 muestra cómo cambia la organización de los elementos pertenecientes a un grupo de nodos, a bajo nivel de zoom los nodos se organizan de forma matricial como se muestra en la imagen de la izquierda, y a alto nivel de zoom, los elementos del grupo se organizan de forma circular.

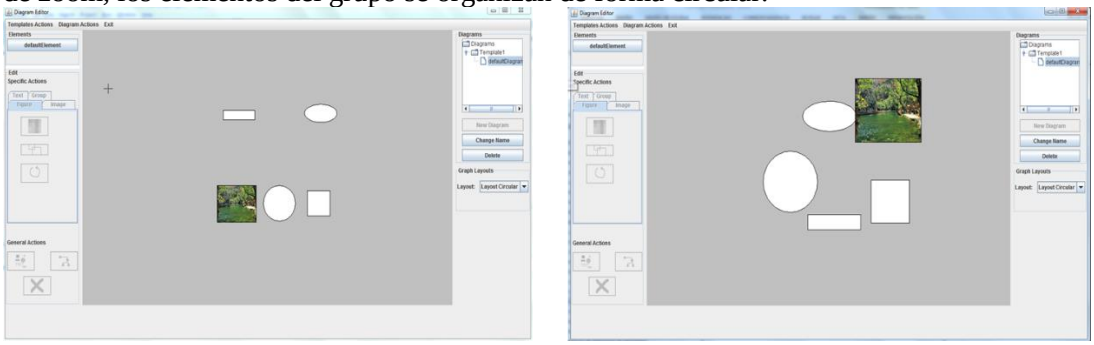


Ilustración 35: Aplicar layout a grupo de nodos.

- Aplicar distintas representaciones de elementos.

Así mismo como en el editor de sintaxis concretas se configuran una o varias representaciones a un elemento, en el editor de diagramas al instanciar este tipo de elementos se puede comprobar que tiene diferentes representaciones haciendo zoom in o zoom out con el botón de scroll del mouse. La Ilustración 36 muestra cómo cambia la representación de elementos de un contenedor de nodos, y en la Ilustración 37 muestra cómo cambia la representación de los elementos pertenecientes a un grupo de nodos.

Para el ejemplo de la Ilustración 36 se presentan dos imágenes, donde cada una de estas hace referencia a una representación diferente de un elemento. En la imagen de la izquierda, hay una elipse en blanco, esta representación se muestra cuando hay muy bajo nivel de zoom, y la imagen de la derecha muestra el contenido de esta elipse alto nivel de zoom. En este ejemplo se puede ver como se aplica a FV-GAIA 2.0 los conceptos de zoom semántico y nivel de detalle.

Para el ejemplo de la Ilustración 33 se presentan dos imágenes también, donde cada una de estas hace referencia a una representación diferente de un elemento. En la imagen de la izquierda, hay un cuadrado con un texto, esta representación se muestra cuando hay muy bajo nivel de zoom, y la imagen de la derecha muestra el contenido de éste cuadrado.

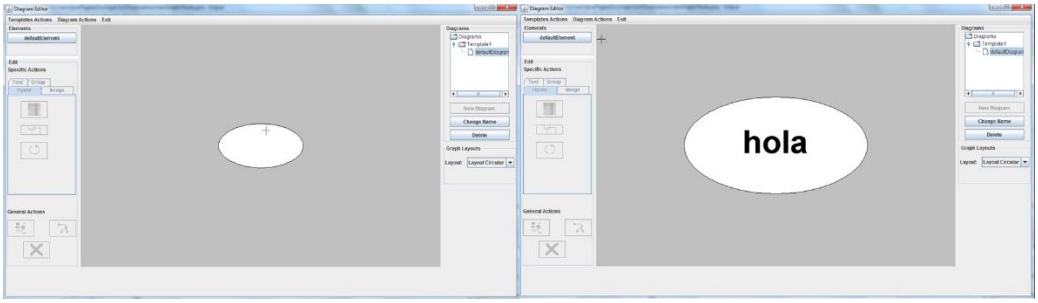


Ilustración 36: Cambio de representación a elementos contenedores

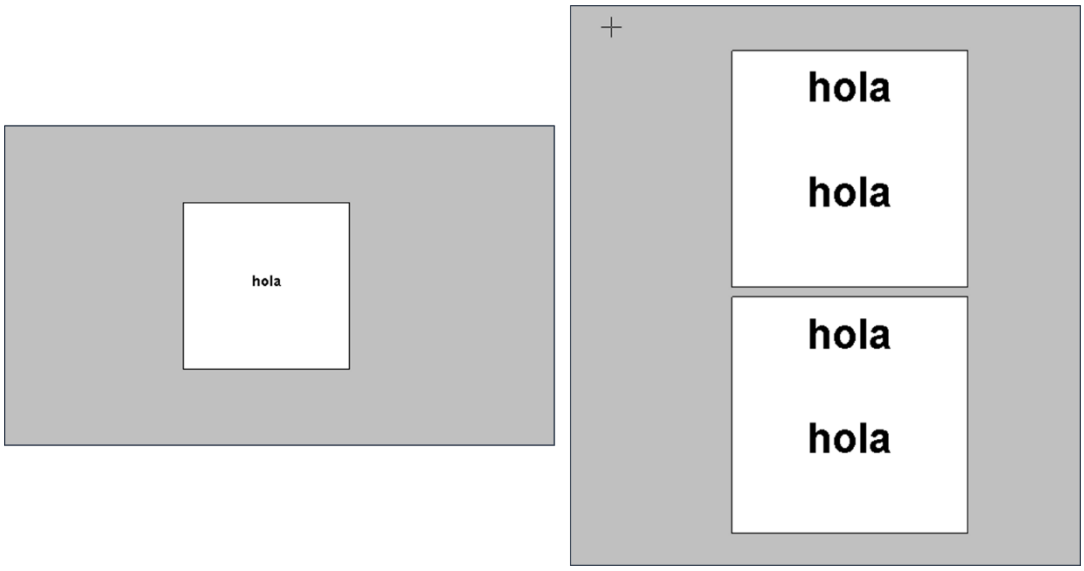


Ilustración 37: Cambio de representación de grupos

- Aplicar layout de grafos.
  Este editor también presenta la funcionalidad de organizar los nodos creados en un documento relacionado por medio de layout de grafos. Para acceder a esta funcionalidad, esta se encuentra a la derecha en un panel de acciones llamado "Graph Layout", como se señala en verde en la Ilustración 38. De acuerdo al tipo de layout seleccionado por el usuario, los nodos que se encuentre creados al momento de la selección cambian de posición y organización, como se ve en la Ilustración 38. Para el caso, se aplicó un layout circular, que organiza todos los elementos del grafo del documento relacionado en forma de circunferencia.

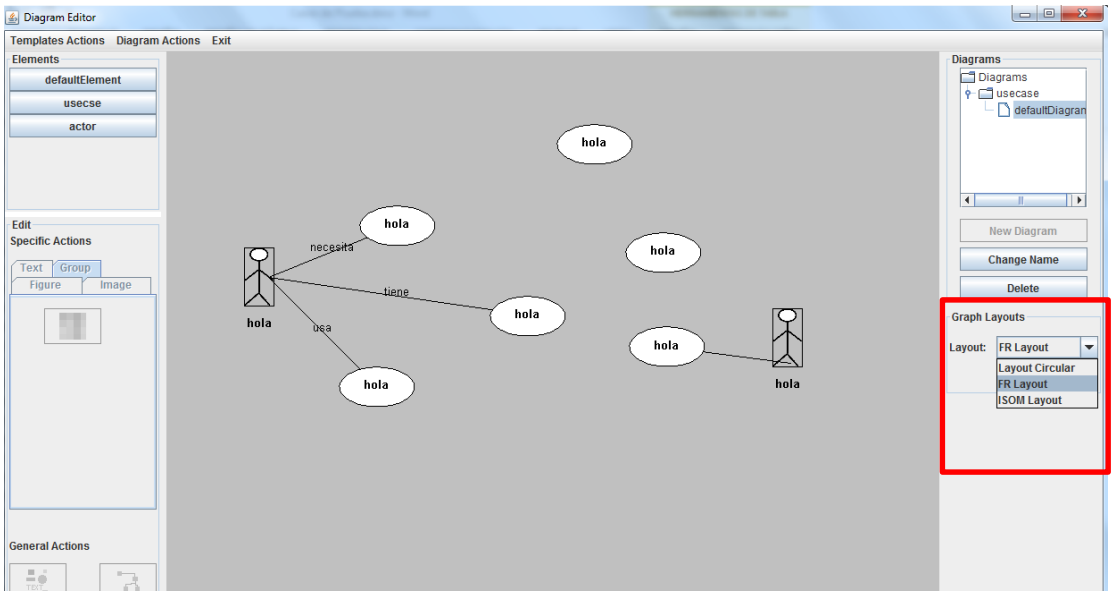


Ilustración 38: Layout de Grafos

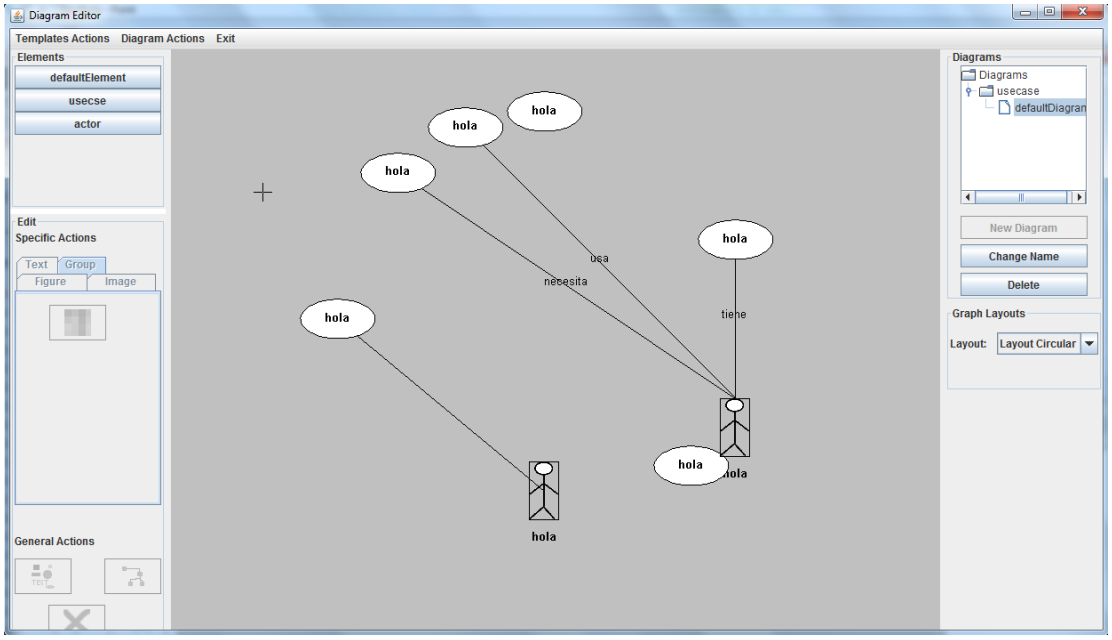


Ilustración 39: Layout de Grafos aplicado.

- Guardar diagramas.
  Este editor de diagrama tiene la posibilidad de abrir y guardar en archivos planos toda la información de los diagramas creados a partir de instancias de plantillas de sintaxis concretas. En estos archivos se guarda toda la estructura del sistema de archivos

del editor de diagramas como lo son: las carpetas de diagramas, sus documentos y elementos visuales que se encuentra instanciados en el editor.

V – Validación y Pruebas de Software

Una vez terminada la especificación, diseño e implementación de FV-GAIA 2.0 se realizó la validación y pruebas del software construido con el propósito de asegurar que todas sus funcionalidades estuvieran cumpliendo según lo establecido en los documentos de especificación de requerimientos. Para tal propósito la validación consiste en la realización de varios tipos de pruebas de software como son las que se describen a continuación.

a) Reporte de Pruebas Unitarias

Las primeras pruebas realizadas al framework desarrollado fueron las pruebas unitarias, las cuales permitieron validar en términos de objetos, instancias y archivos planos si la información se esté guardando de forma correcta y consistente. Estas pruebas se realizaron a dos componentes de FV-GAIA 2.0 en los dos editores visuales; el primero fue el componente de persistencia y el segundo el componente que guarda objetos en memoria. Estas pruebas unitarias se crearon y automatizaron por medio de JUnit [37]. Como resultado de estas pruebas se realizaron 4 pruebas donde todas fueron exitosas, dos por cada editor visual a sus componentes respectivos de persistencia y objetos en memoria.

b) Casos de Pruebas

Para la verificación, validación y prueba de las diferentes funcionalidades de FV-GAIA 2.0 en las pruebas de sistema y aceptación se generaron varios casos de pruebas que permitieron asegurar que los requerimientos establecidos para el diseño y especificación de este aplicativo se están cumpliendo. En la Tabla 2 se presenta los casos establecidos para probar las funcionalidades de cada uno de los editores visuales.

Editor de Sintaxis Concretas:	• Creación de plantillas de Sintaxis Concretas. • Creación de documentos de plantillas de Sintaxis Concretas. • Creación de diferentes nodos. • Editar nodos simples. • Editar nodos complejos. • Traslación de nodos simples. • Aplicar transformaciones geométricas a nodos. • Creación y manipulación de nodos contenedores. • Creación y manipulación de aristas contenedoras. • Aplicar layouts a nodos contenedores. • Agregar layouts a nodos contenedores. • Crear y trasladar grupos de nodos. • Deshacer grupo de nodos. • Aplicar layouts a grupos de nodos. • Agregar layouts a grupos de nodos.

	• Pegar nodos y trasladar nodos pegados. • Permitir que un nodo pueda tener diferentes representaciones de acuerdo al nivel de zoom. • Guardar archivos de plantillas de sintaxis concretas. • Abrir archivos de plantillas de sintaxis concretas.
Editor de Diagramas:	• Editar archivos de plantillas de sintaxis concretas. • Cargar archivos de plantillas de sintaxis concretas. • Creación de documentos de diagramas. • Instanciación de plantillas de sintaxis concretas (Nodos). • Instanciación y traslación de plantillas de sintaxis concretas (Aristas). • Trasladar instancias de plantillas de sintaxis concretas. • Aplicar transformaciones geométricas a instancias de plantillas de sintaxis concretas. • Instanciar y trasladar grupos de plantillas de sintaxis. • Crear y trasladar grupos de plantillas de sintaxis concretas. • Deshacer grupo de plantillas de sintaxis concretas. • Instanciar y trasladar contenedores de plantillas de sintaxis. • Crear y trasladar contenedores de plantillas de sintaxis concretas. • Aplicar distintos layouts a nodos contenedores y cambiar el zoom de la cámara. • Aplicar distintos layouts a nodos agrupados y cambiar el zoom de la cámara. • Instanciar plantillas de sintaxis concretas que cambian su representación de acuerdo al nivel de zoom. • Organizar plantillas de sintaxis concretas en layout circular u ortogonal. • Guardar archivos de diagramas de nodos. • Abrir archivos de diagramas de nodos.

Tabla 3: Casos de Pruebas

c) Reporte de Pruebas de Sistema

Luego de las pruebas unitarias y guiado por los casos de pruebas presentados con anterioridad se realizaron las pruebas de sistema donde se probaron cada uno de los casos de forma minuciosa, siguiendo una serie de pasos y tomando pantallazos en cada uno de los pasos para asegurar la realización de las diferentes pruebas. En total se realizaron 35 pruebas de sistema relacionadas con los dos editores visuales donde fueron exitosas 30. Para ilustra las pruebas de sistema realizadas, a continuación se presenta un escenario de pruebas y su respectiva prueba.

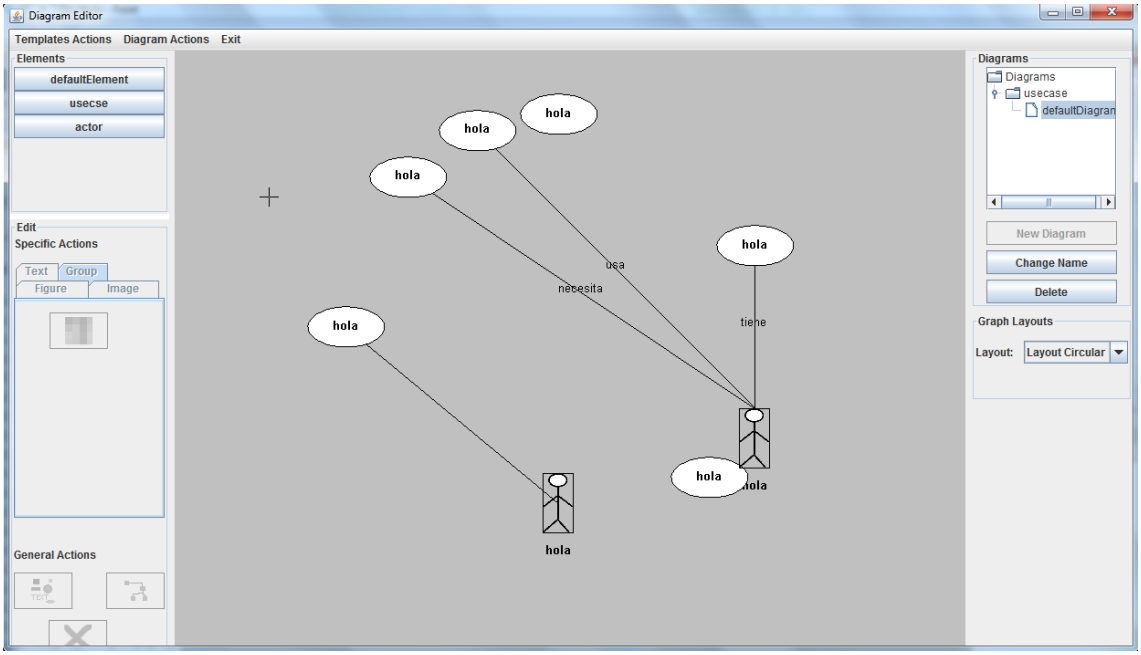
Id:	CP035	***Nombre:***	Organizar plantillas de sintaxis concretas en layout circular u ortogonal.

Requerimiento(s) Relacionado	R020 y R021.
Aplicación Relacionada	Editor de Diagramas.
Objetivo en Contexto	Seleccionar un layout de grafo y aplicar el layout seleccionado a todas las plantillas de sintaxis concretas instanciadas en el editor de diagramas.
Pre-Condiciones	El editor de diagramas debe estar en ejecución y se debió haber cargado un archivo de plantillas de sintaxis concretas. También se debe haber instanciado al menos cinco plantillas de sintaxis concretas con anterioridad.
Post-Condiciones	De acuerdo al layout de grafos seleccionado, todas las instancias de plantillas de sintaxis concretas creadas en el editor de diagramas cambian de posición inmediatamente.
Resultado esperado	Visualización en el lienzo de trabajo de cómo cambia la organización de todas las instancias de plantillas de sintaxis concretas creadas.

Flujo básico					
	Usuario		*Sistema*	*Resultado*	*Comentario*
1	Situarse en el panel de layout de grafos “Graph Layout”.			**Prueba Exitosa.**	
2	Seleccionar un Layout a aplicar.	3	El editor procesa la orden de cambio de layout de acuerdo al tipo de layout seleccionado.		
		4	El editor cambia de posición las plantillas de sintaxis concretas instanciadas junto con sus conexiones a una nueva posición de acuerdo al	**Prueba Exitosa.**	

			tipo layout seleccionado.		

Prueba Realizada
Ejecución del Editor de Diagramas    Paso 2 Flujo Básico



d) Reporte de Pruebas de Aceptación

Por último y para terminar las validaciones y pruebas de software, se realizaron las pruebas de aceptación por parte del director del trabajo de grado que para este caso también tiene el rol de cliente, esta pruebas también están basadas en los casos de prueba presentados con anterioridad. En estas pruebas también se probaron cada uno de los casos, siguiendo los pasos de las pruebas de sistema y tomado pantallazos para asegurar la realización de las mismas.

VI – Conclusiones, Recomendaciones y Trabajos Futuros

a) Conclusiones

Como conclusiones de este trabajo de grado, se enumeran las siguientes:

- Luego de un año de trabajo, FV-GAIA 2.0 satisface 30 de los 32 requerimientos establecidos, divididos entre los dos editores.
- FV-GAIA 2.0 es un Framework que cumple con las expectativas propuestas en términos funcionales y no funcionales, ya que está implementado con buenas prácticas de ingeniería de software lo que permite que sea usable y se puedan realizar actualizaciones sobre este.
- La generación de sintaxis concretas en interfaces aumentables es posible por medio de una interfaz de usuario como por medio de código fuente.

- FV-GAIA 2.0 es un framework más robusto, flexible y escalable en comparación a FV-GAIA, ya que no solo tiene muchas más funcionalidades sino que también se puede aplicar para desarrollar cualquier tipo de diagrama visual.

b) Recomendaciones

Luego de la realización de los diferentes componentes de la arquitectura de FV-GAIA2.0, y de ver el gran uso de las listas y mapas de objetos de Java se puede ver cómo es recomendable partir un problema en varios, y atacarlos desde otros puntos de vista. Se recomienda a estudiantes y desarrolladores que no se esmeren y esfuercen por solucionar un problema por un solo lado, sino que mire primero a su alrededor, salgan del bosque para tener una visual de todo el problema y analicen las diferentes soluciones, un problema tiene demasiadas soluciones.

A los desarrolladores que piensen continuar o utilizar este framework se recomienda que tengan muy buenos conocimientos sobre patrones de diseño y mucha disposición para investigar a fondo cada uno de las librerías por aparte y dentro de java, ya que es de mucha ayuda para realizar una buena implementación de este tipo de aplicativos.

c) Trabajos Futuros

Tomando este proyecto como punto de partida para otros trabajos de grado o proyectos, el paso a seguir sería refinar los requerimientos relacionados con escalar y rotar tanto primitivas gráficas como plantillas de sintaxis concretas.

Luego de realizada esta pequeña modificación al proyecto, a continuación se describen los posibles trabajos futuros basados en el funcionamiento y aplicación del framework FV-GAIA2.0:

- Se debería implementar el modulo que queda faltando en la arquitectura de Zoo-MEnv [3], que es el Modelo. Esto hace referencia a toda la parte que tiene que ver el administrador de modelos, que se refiere a administrar modelos y las sintaxis concretas haciendo uso de EMF para lograr independencia entre la generación de código a partir de un archivo Ecore, haciendo uso de las sintaxis concretas desarrolladas en FV-GAIA 2.0.
- Otro trabajo futuro podría ser, tomar un generador de código fuente basado en sintaxis concretas y utilizar este framework para generar una herramienta CASE completa donde no solo sea un visualizador de elementos de diagramas y diagramas en general, sino que también permita generar código fuente de los diagramas creados. Luego de realizar la implementación anterior, FV-GAIA 2.0 no se vería como una pequeña herramienta de modelado, sino como otra opción que pueda competir con grandes marcas como Rational Rose o Enterprise Architect [38] [39].
- También con los editores que presentó FV-GAIA 2.0 se podría realizar un estudio de usabilidad donde se pruebe de forma exacta y concreta la real usabilidad que tiene estos editores en cuanto a usuarios finales y como se podrían mejorar teniendo en cuenta los usuarios finales y el propósito del framework.

- Los resultados e insumos de este trabajo de grado se podrían utilizar para generar un artículo de investigación donde no solo se describa la problemática y el proceso de ingeniería de software que se llevó a cabo sino los resultados obtenidos.

6. ANEXOS

Anexo 1. Glosario

Este anexo es un documento que presenta todos los conceptos y bibliografía utilizada durante el trabajo de grado, durante el desarrollo de este documento y los documentos asociado al trabajo de grado.

Anexo 2. Documento Visión

Este anexo presenta toda la especificación del producto de software junto con su funcionalidad, su mercado objetivo y tiene asociado un solo documento que presenta toda la especificación de FV-GAIA 2.0 como producto de software.

Anexo 3. Documento SRS

Este anexo se refiere a todo el proceso de análisis que se realizó a los requerimientos especificados por el director del trabajo de grado y presenta cuatro documentos asociados. El primero presenta toda la documentación del proceso de análisis de requerimientos, el segundo presenta la evaluación de calidad que se realizó a cada uno de los requerimientos, el tercero presenta la matriz de trazabilidad con la cual se siguió cada uno de los documentos a través del código fuente y el ultimo es el documento de medición de requerimientos que presenta el estado de cada uno de los requerimientos durante el desarrollo de la aplicación y en su culminación.

Anexo 4. Documento SDD

Este anexo se refiere a todo el proceso que se llevó a cabo para realizar el diseño de la aplicación y presenta varios diagramas asociados donde se especifica la arquitectura de software que caracteriza a FV-GAIA 2.0.

Anexo 5. Reporte de Pruebas

Este anexo se refiere a todo el reporte de pruebas que se llevó a cabo luego de la implementación de FV-GAIA 2.0 como producto de software el cual comprende la documentación de los reportes de pruebas Unitarias, de Sistemas y de Aceptación

Anexo 6. Manual de Usuario Desarrollador

Este anexo se refiere a la guía realizada de cada una de las funcionalidades del framework paso a paso donde se detalla lo que se debe hacer y cómo debe ser el uso de FV-GAIA 2.0 por interfaz gráfica de usuario y por código fuente.

7. REFERENCIAS

[1] Nipun Tomar, «Computer Aided Software Engineering Tools (CASE)», 26-04-2011. [En línea]. Disponible en: http://www.c-sharpcorner.com/UploadFile/nipuntomar/computer-aided-software-engineering-tools-case/. [Accedido: 26-oct-2014].

[2] Steeve Callan, «Making the Case for CASE Tools», Database Journal, 10-ago-2005. [En línea]. Disponible en: http://www.databasejournal.com/features/oracle/article.php/3525621/Making-the-Case-for-CASE-Tools.htm. [Accedido: 26-oct-2014].

[3] Jaime A. Pavlich M, Hernan D. Veliz Q, steven A. Demurjian and Laurent D. Michel, "Un ambiente de meta-modelamiento y visualización basado en el paradigma de zoomable user interfaces".

[4] Lance G. y Benjamin B. B, "Zoomable user interfaces as a medium for slide show presentations", Information Visualization, 2002.

[5] Fréderic Fondement, «Concrete Syntax Definition for Modeling Languages», 2007. [En línea]. Disponible en: http://infoscience.epfl.ch/record/111344/files/EPFL_TH3927.pdf. [Accedido: 27-oct-2014].

[6] T.T.A. Combs and B.B. Bederson, "Does zooming improve image browsing?", Proceedings of the fourth ACM conference on Digital libraries, Berkeley, California, United States: ACM, 1999, pp. 130-137.

[7] Federico Rodriguez B, "FVGAIA: Framework para la Visualización de Grafos de Software basado en Interfaces Aumentables", Pontificia Universidad Javeriana, Trabajo de grado, 2012.

[8] Diagram Definition, consulta realizada 18/06/2014, [En línea] Disponible en: http://www.omg.org/spec/DD/.

[9] Scott W. Ambler, "The Agile Unified Process", http://www.ambysoft.com/unifiedprocess/agileUP.html, Ambysoft.

[10] Mauricio Alberto Sanchez Franco, "Ambiente de visualización y manipulación de grafos con ayuda de interfaces aumentables". Propuesta de Trabajo de Grado. Pontificia Universidad Javeriana, 2015.

[11] C. Geiger, H. Reckter, R.Dumitrescu, S.Kahl, and J. Berssenbrügge, "A Zoomable User Interface for Presenting Hierarchical Diagrams on Large Screens", Human-Computer Interaction, Part II, HCII 2009, 2009.

[12] M. Bennett and F.Cummins, "ORRIL: A Simple Building Blocks Approach to Zoomable User Interface", Media Lab Europe, University College Dublin, and the HEA (HigherEducation Authority) in Ireland.

[13] Na Liu, John Hosking and John Grundy, Integrating a Zoomable User Interfaces Concept into a Visual Language Meta-tool Environment. [En línea]. Disponible en: https://www.cs.auckland.ac.nz/~john-g/papers/hcc2004_2.pdf. [Accedido: 31-ago-2015].

[14] «Piccolo2D» [En línea]. Disponible en: http://piccolo2d.mro.name/learn/grapheditor.html. [Accedido: 05-ago-2015].

[15] «ZVTM – Zoomable Visual Transformation Machine, consulta realizada 10/09/2011, [En línea] Disponible en: http://zvtm.sourceforge.net/.

[16] «Eagle Mode» [En línea]. Disponible en: http://eaglemode.sourceforge.net. [Accedido: 11-sep-2011].

[17] «Clutter» [En línea]. Disponible en: http://www.clutter-project.org/.[Accedido: 11-sep-2011].

[18] «CZui» [En línea]. Disponible en: http://czui.sourceforge.net/.[Accedido: 11-sep-2011].

[19] Y. SUGIMORI, K. KUSUNOKI , F. CHO & S. UCHIKAWA, «Toyota production

system and Kanban system Materialization of just-in-time and respect-for-human system», International Journal of Production Research. [En línea]. Disponible en: http://www.tandfonline.com/doi/pdf/10.1080/00207547708943149. [Accedido: 31-ago-2015].

[20] ZGRViewer a GRaphViz/DOT Viewer, consulta realizada 31/01/2012, [En línea] Disponible en: http://zvtm.sourceforge.net/zgrviewer.html.

[21] Jaime A. Pavlich, ZooMEnv-2.0 Concrete Syntax Requirements, presentación power point.

[22] Zoomable User Interface, consulta realizada 31/01/2012, [En línea] Disponible en: http://www.usabilitypost.com/2009/02/09/zoomable-user-interfaces/, enero 31/2012.

[23] Instituto nacional de estadística e informática, "Herramientas Case", consulta realizada 20/08/2012, [En línea] Disponible en: http://www.inei.gob.pe/biblioineipub/bancopub/Inf/Lib5103/Libro.pdf.

[24] Groovy, consulta realizada 28/09/2015, [En línea] Disponible en: http://www.groovy-lang.org/

[25] EMF, consulta realizada 28/09/2015, [En línea] Disponible en: https://eclipse.org/modeling/emf/.

[26] Karl E. Wiegers, First Things First: Prioritizing Requirements, consulta realizada 04/02/2012, [En line] Disponible en: http://www.processimpact.com/articles/prioritizing.html.

[27] I. Sommerville, Software Engineering, Ninth Edition. New York: Pearson Education Inc. 2010.

[28] Matriz de trazabilidad de requerimientos, , consulta realizada 05/02/2012, [En línea] Disponible en: http://webcache.googleusercontent.com/search?q=cache:V7vYWBg0DV8J:www2.cdc.gov/cdcup/library/templates/CDC_UP_Requirements_Traceability_Matrix_Template.xls+Requirement+Traceability+Matrix&cd=2&hl=es&ct=clnk&gl=co.

[29] Requirements Traceability, , consulta realizada 05/02/2012, [En línea] Disponible en: http://www.gatherspace.com/static/requirements_traceability.html.

[30] StartUML, consulta realizada 14/11/2015, [En línea] Disponible en: http://staruml.sourceforge.net/en/.

[31] NetbeansIDE, consulta realizada 14/11/2015, [En línea] Disponible en: http://netbeans.org/community/releases/55/index.html.

[32] [Dia, consulta realizada 14/11/2015, [En línea] Disponible en: http://projects.gnome.org/dia/.

[33] JDeveloper, consulta realizada 14/11/2015, [En línea] Disponible en: http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html.

[34] Together, consulta realizada 14/11/2015, [En línea] Disponible en: http://www.borland.com/us/products/together/index.aspx.

[35] Sean Boyd, Mark D'Adamo, Christopher Horne, Nolan Kelly, David Ryan and Nairn Tsang, SOFTWARE ARCHITECTURAL STYLES, 2013, [En línea] Disponible en: http://kremer.cpsc.ucalgary.ca/courses/seng403/W2013/papers/04ArchitectureStyles.pdf.

[36] Trygve Reenskaug and James O. Coplien, The DCI Architecture: A New Vision of Object-Oriented Programming, consulta realizada 7/10/2015, [En línea] Disponible en: http://www.artima.com/articles/dci_vision.html.

[37] JUnit, consulta realizada 24/04/2016, [En línea] Disponible en: http://junit.org/junit4/.

[38] Rational Software, consulta realizada 14/11/2011, [En línea] Disponible en: http://www-01.ibm.com/software/rational/.

[39] Enterprise Architect, consulta realizada 14/11/2011, [En línea] Disponible en: http://www.sparxsystems.com/products/ea/index.html

[40] OpenGL, consulta realizada 05/02/2016, [En línea] Disponible en: http://www.opengl.org.

[41] VRML, consulta realizada 05/02/2016, [En línea] Disponible en: http://mheller.com/vrmlbasics.htm.

[42] Eduardo Montenegro León y Daniel Leonardo Rico Hernández, "ZOOMIT++: Framework para visualizar diagramas de software mediante Zoomable User Interfaces y tecnología Multi-Touch", Pontificia Universidad Javeriana, Trabajo de grado, 2015.