CIS1510AP01

ZOOMTI: *Framework* para visualizar diagramas de *software* mediante *Zoomable User Interfaces* y tecnología *Multi-Touch*

Eduardo Montenegro León
Daniel Leonardo Rico Hernández

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTÁ, D.C.
2015

CIS1510AP01

ZOOMTI: *Framework* para visualizar diagramas de *software* mediante *Zoomable User Interfaces* y tecnología *Multi-Touch*

Autor(es):

Eduardo Montenegro León
Daniel Leonardo Rico Hernández

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

Director

Jaime A Pavlich-Mariscal

Jurados del Trabajo de Grado

Lina María Consuelo Franky de Toro

Leonardo Flórez-Valencia

Página web del Trabajo de Grado

http://pegasus.javeriana.edu.co/~CIS1510AP01

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTÁ, D.C.
Mayo, 2015

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS

Rector Magnífico

Jorge Humberto Peláez Piedrahita, S.J.

Decano Facultad de Ingeniería

Ingeniero Jorge Luis Sánchez Téllez

Director de la Carrera de Ingeniería de Sistemas

Ingeniero Germán Alberto Chavarro Flórez

Director Departamento de Ingeniería de Sistemas

Ingeniero Rafael Andrés González Rivera

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

Agradecimientos de Daniel Rico

Agradezco a mi familia por siempre brindar su apoyo incondicional ante cada una de las decisiones que he tomado en el largo camino que es la vida. Cada consejo recibido es y seguirá siendo igual de importante a lo aprendido durante estos años de carrera.

Agradezco en especial a mi hermana Claudia por depositar su confianza en mí y volver realidad el sueño impensado de estudiar en la Universidad Javeriana. Su apoyo de comienzo a fin, me permite estar a días de graduarme como ingeniero, estaré en deuda toda la vida por este favor. De igual manera a mi hermana Sandra por el apoyo en la elaboración del trabajo de grado, gracias a ella se logró darle un toque de distinción a *ZoomTI++*.

Agradezco al Capítulo Javeriano ACM por mostrarme la verdadera manera de vivir la universidad, "la actitud es la clave" es más que una frase representativa, es un estilo de vida. Una gran parte de mi crecimiento personal y profesional en estos años fue gracias a compartir con personas tan completas. Desde la primera maratón que ayudé como organizador, hasta la última reunión como integrante de la junta directiva, aprendí algo cada día.

Agradezco a nuestro director de trabajo de grado por depositar su confianza en nosotros como equipo de trabajo, por observar nuestras capacidades y ayudarnos a definir una meta que fuera digno desafío para terminar la vida universitaria. De igual manera por estar siempre en la mejor disposición para liberarnos de dudas y guiarnos a encontrar nuestras propias respuestas.

Finalmente una pequeña mención a Catalina por la ayuda en la elaboración del icono, es agradable saber que hay personas las cuales brindan apoyo en todo momento.

Agradecimientos de Eduardo Montenegro

Primero, quiero agradecerle a mi Mamá Gloria Inés por ser apoyo incondicional en cada aspecto de mi vida, en especial el de mi carrera profesional, su espíritu, fuerza y determinación son los que hoy me tienen cerca de mi grado. Este título, es de ella.

A mi familia por ser pieza fundamental en mi desarrollo como persona, juntos logramos cumplir nuestras metas, y hoy son ellos los que hacen posible que termine mi carrera.

A Catalina Jaime y Sandra Rico por su ayuda, gracias a ellas fue posible el desarrollo de un trabajo de grado de calidad.

Por ultimo a nuestro director de trabajo de grado Jaime Pavlich que por medio de su conocimiento, no solo ayudo a la finalización y desarrollo de este trabajo de grado, sino a mi formación como profesional.

CONTENIDO

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

ABSTRACT

The way in which information is displayed, is an important part of a software engineering project; if there is incomplete or confuse access to the information, it can generate on later stages different problems and inconsistencies. This degree project developed a framework, called ZoomTI++, able to visualize software diagrams by using Zoomable User Interfaces and to navigate through the information by using a multi-touch device. ZoomTI was developed on C++ in order to reach high performance towards diagrams with a big number of elements.

RESUMEN.

La forma en la que se visualiza la información, es una parte importante en un proyecto de ingeniería de *software*; esto debido a que con un acceso a la información incompleto o confuso puede generar en etapas posteriores varios problemas e inconsistencias. Este trabajo de grado desarrolló un *framework,* llamado ZoomTI++, capaz de visualizar diagramas de software, por medio de interfaces aumentables (*Zoomable User Interfaces*) y capaz de navegar a través de la información, por medio del uso de un dispositivo *multi-touch.* ZoomTI++ fue desarrollado en C++ para alcanzar un alto rendimiento ante diagramas con alto número de elementos.

INTRODUCCIÓN

La presente memoria recoge todo el proceso que fue el desarrollo de la propuesta de trabajo de grado "ZoomTI++: *Framework* para visualizar diagramas de software mediante *Zoomable User Interfaces* y tecnología *Multi-Touch*". En los párrafos posteriores se explica la manera en la que se encuentra distribuido y organizado el documento.

La primera sección busca explicar el contexto sobre el cual se desarrolló el trabajo de grado; la problemática que generó el trabajo, la razón por la que era importante atacar dicha problemática y de qué manera se buscó brindar una solución a esa situación, incluyendo la metodología seguida para alcanzar la solución. La segunda sección explica soluciones relacionadas hacia la problemática y porque la solución desarrollada en este trabajo de grado tiene un elemento diferenciador hacia ellas. De igual manera se presentan conceptos importantes para la comprensión de la solución elaborada.

La tercera sección muestra las consideraciones tomadas antes del desarrollo de la solución, esto incluye los requerimientos del *framework*, con sus respectivas justificaciones, y labores de priorización. De igual manera se incluye los análisis realizados para la selección de herramienta base. La cuarta sección hace referencia al diseño de la aplicación para cumplir los requerimientos y haciendo uso del análisis para la selección de la herramienta base. Se presenta la arquitectura del *framework* final.

La quinta sección se enfoca en lo que fue el desarrollo de ZoomTI++, explicando la manera en la que fue seguida la metodología y presentando el producto final a través de imágenes. La sexta sección se centra en la evaluación del *framework* desarrollado a partir de las pruebas definidas. La séptima sección, a partir de los resultados de las secciones anteriores, concluye acerca de lo alcanzado con ZoomTI++ desde diferentes campos y plantea un escenario futuro para el continuo crecimiento del *framework*.

Las dos últimas secciones son dedicadas a abrir la posibilidad hacia más información en los temas desarrollados en la presente memoria: En primer lugar la sección de referencias incluye enlaces a documentos, paginas, entre otros con mayor información para profundizar en las temáticas. La sección final muestra los anexos importantes para mayor comprensión de lo expuesto a la memoria, incluyendo enlaces a documentos con análisis más completos.

I - DESCRIPCIÓN GENERAL

1. Oportunidad, Problemática, Antecedentes

La forma en la que se visualiza la información es una parte importante en un proyecto de ingeniería de software, esto debido a que con un acceso a la información incompleto o confuso puede generar en etapas posteriores varios problemas e inconsistencias. [5] Un problema que se puede presentar es cuando en un modelo de software la cantidad de información es extremadamente alta [6]. De manera opuesta, información clara y completa ayudará que en etapas futuras el proyecto marche de la mejor manera. [2]

Existe una gran variedad de herramientas para la visualización de modelos, como es el caso de *Graffletopia* [16] u *OmniGraffle* [17] pero que no se usan en gran medida en proyectos de ingeniería de software; existen a su vez las herramientas *Computer-aided Software Engineering* (CASE) [8][9] que permiten, además de la visualización de los modelos, diferentes funcionalidades que apoyan al proceso de desarrollo de un proyecto de software [7]. Sin embargo, la mayor parte de estas trabajan de la misma manera. Son herramientas cuya interacción se da en gran parte a través de barras de desplazamiento y a partir de un sistema de navegación de archivos integrado a las aplicaciones, generalmente representado en forma de árbol jerárquico, que obligan al usuario a tener cuidado en la visualización de modelos relacionados. Este tipo de herramientas pueden mejorar su funcionamiento, si lograran afinar el razonamiento espacial del usuario [1].

Para un ingeniero civil se puede tratar del desplazamiento espacial sobre los planos de un edificio que permiten representar con un nivel alto de fidelidad una abstracción de este, creando una sensación de desplazamiento real sobre el edificio terminado. Un ejemplo paralelo en un proyecto de ingeniería de software se ve reflejado en un diagrama de casos de uso, en dónde para cada uno de los casos de uso se puede tener un diagrama de secuencia relacionado, sólo visible dependiendo del nivel de detalle que se quiera ver, en otras palabras, al hacer *zoom out* se logre ver sólo el diagrama de casos de uso, y al hacer *zoom in* en cualquier caso de uso poder ver su diagrama de secuencia. La recreación de este efecto en una herramienta de visualización permite al desarrollador explorar algunos rincones del modelo de forma espacial.

Debido a esta problemática se ha buscado una manera en la cual el uso de las herramientas *CASE* [8][9] sean lo suficientemente claras en el uso del razonamiento espacial con el usuario. Esto ha implicado que en los últimos años hayan aparecido más componentes de apoyo a las herramientas *CASE* [8][9] que usan el principio de las *Zoomable User Interfaces* (ZUI), abriendo nuevas posibilidades en la navegación de modelos al permitir una interacción más intuitiva y transparente para el usuario. [3]

La necesidad de este tipo de componentes no es algo totalmente nuevo; debido al aumento de la complejidad en los requerimientos con el paso del tiempo, la necesidad de adaptar cambios en un modelo de forma rápida, transparente y con el menor impacto posible, entre otras más [1] abrió paso a buscar esas novedosas aproximaciones. De estas aproximaciones se termina generando una serie de requerimientos que toda herramienta apoyada en *ZUI* debe cumplir. Sin embargo, en la actualidad la realidad es que a pesar de que las necesidades han sido claramente

identificadas, no hay trabajos extensos en el tema que se encarguen de aplicar estos conocimientos obtenidos, no obstante hay aproximaciones realizadas al interior de la Pontificia Universidad Javeriana, bajo la dirección del ingeniero Jaime Pavlich, las cuales fueron realizadas en *Java* [18] y en *HTML5* (*HyperText Markup Language*) [19], de estos trabajos de grado nació una oportunidad de hacer uso de esta situación, abriendo posibilidades de aplicar estos conocimientos junto con el uso de un lenguaje nuevo y relacionarlos con otros campos como se presenta a continuación.

Al hacer uso del razonamiento espacial y la manera en la que este permite obtener información del modelo de una manera automática, se puede mejorar la productividad del usuario. Sin embargo, al ser una aproximación tan reciente todavía se encuentra muy inmadura; las herramientas *CASE* [8][9] apoyadas por ZUI carecen de un gran número de funciones clásicas en comparación con herramientas tradicionales y al ser estas últimas tan sencillas, las herramientas apoyadas por ZUI son usadas por pocas personas debido a sus limitaciones versus sus beneficios en relación a herramientas más completas y tradicionales [1].

De esta manera se alcanzó un punto complicado, se poseen herramientas con una capacidad lo suficientemente amplia para brindar nuevos horizontes en la abstracción de información, sin embargo hoy día aún no son capaces de cubrir las nuevas necesidades de los usuarios [10].

Adicionalmente, a las ventajas de los componentes *ZUI* que apoyan a las herramientas *CASE* [8][9] hay otra tecnología cuya interacción con estos componentes aún no ha sido explorada, es el uso de tecnología *multi-touch*. Este tipo de tecnología permite al usuario controlar el sistema de una manera diferente, dejándolo interactuar con la dimensionalidad del modelo, lo cual en relación con ZUI, puede abrir un universo de posibilidades que para el usuario resultan intuitivas, como es el uso del zoom semántico a partir del uso de los dedos. [4]

Por una parte, la persistencia en proyectos de ingeniería de software es necesidad intrínseca sin importar la magnitud del mismo, por consiguiente, en los proyectos que involucren modelos de software se vuelve una tarea que tiene que ser realizada por una herramienta de apoyo *CASE*[8][9]. Mediante el uso del lenguaje de programación *C++* [16] se puede tener la persistencia de los modelos y la visualización de los mismos de una manera no sólo eficaz sino eficiente, lo cual brindaría gran beneficio al usuario que cuente con un modelo de gran magnitud. Por otra parte, como se mencionaba anteriormente, no se ha abordado hasta el momento un proyecto con estas características desarrollado en este lenguaje, pudiendo generar una visualización mucho más fluida en comparación de las aproximaciones hechas hasta ahora.

Combinando estos factores, junto al uso del lenguaje de programación C++ [16] y con la ayuda de los requerimientos derivados del trabajo de grado FVGAIA [24], se creó la posibilidad de buscar una nueva aproximación para la interpretación de diagramas por medio del *framework ZoomTI++*.

1.1. Formulación del problema que se resolvió

¿Cómo apoyar la creación de una herramienta de navegación y visualización en modelos de *software*, a través de *Zoomable User Interfaces*, tecnologías *multi-touch* y el uso del lenguaje de programación C++?

1.2. Justificación del problema

La importancia de la optimización tanto de los componentes de navegación como los de visualización basados en *ZUI* todo enfocado en herramientas *CASE* [8][9] radica en que este enfoque permite abrir oportunidades previamente desconocidas [1], hecho reforzado por el uso de tecnologías *multi-touch*, persistencia de datos y en combinación del lenguaje de programación *C++*, todo en la constante búsqueda de permitir al usuario de usar sus capacidades a nuevos límites.

Un conjunto de librerías para apoyar la creación de una nueva herramienta o en otras palabras un *framework*, llamado *ZoomTI++*, facilita la comprensión de modelos de software mediante el uso de las manos del usuario, al estar usando una pantalla táctil. [11]. Al usar sus manos para navegar alrededor de un diagrama le permite realizar diferentes acciones no solo limitadas a realizar *zoom in* y *zoom out* el usuario puede llevar su razonamiento a nuevos límites, siempre respetando la integridad del modelo.[4][12] Se creó un *framework (ZoomTI++)* que permite la interacción en tiempos de espera cortos. [13] Por ejemplo, al estar observando un diagrama de clases, basado en el nivel de zoom, la representación en pantalla para el usuario es variable: Se puede pasar de solamente observar el nombre de las clases (a un nivel lejano) a ver en detalle los métodos y atributos de la clase (a un nivel cercano).

Al poner una herramienta basada en este componente a disposición del usuario, el cual es esperado que cuente con conocimientos básicos ya sea como arquitecto de *software* o ingeniero de *software*, podría beneficiarse mediante el uso de razonamiento espacial, ya que le permitiría mejorar su productividad y realizar labores que previamente no eran intuitivas, esto le ayudaría a enfocarse en otros aspectos de los modelos [13] los cuales no se consideraban para mejoras debido a que el tiempo usado para lograr el nivel de abstracción se ahorraría con el uso de una nueva herramienta apoyada por el nuevo *framework* que se encarga automáticamente de este aspecto. [4]

Como factor complementario, se realizó la creación del *framework* en el lenguaje de programación *C++* [16], que permite además de visualizar el modelo con los componentes ZUI y *multi-touch*, cuenta con la capacidad de leer diferentes modelos persistidos, en donde el usuario pueda recibir la información necesaria del modelo en un periodo de tiempo corto.

Un usuario beneficiado por el *framework* ZoomTI++ podría mejorar su productividad, como ya se mencionó, es decir, que las personas que directamente están interesadas en los resultados de su trabajo se podrían beneficiar, aunque es normal que estas personas no comprendan el beneficio de usar la herramienta salvo en casos específicos donde el cliente posee conocimiento en este campo.[4]

1.3. Impacto Esperado

A mediano y largo plazo se espera que *ZoomTI++* pase de ser un *framework* a una herramienta con todas las funcionalidades de una herramienta tradicional para el uso posterior de personas involucradas en proyectos de Ingeniería de software que vean en el razonamiento espacial una mejor manera de modelar sus proyectos.

2. Descripción del Proyecto

2.1. Objetivo general

Desarrollar un *framework* en *C++* que use tecnología *multitouch* y *Zoomable User Interfaces* para visualización y navegación de modelos de software.

2.2. Objetivos específicos

- Extender la especificación de requerimientos del trabajo de grado FVGAIA [24] y de la herramienta *ZooMEnv* con requerimientos asociados a tecnologías *multi-touch*.
- Diseñar un *framework* usando tecnología *multi-touch* y *ZUI* que cuente con lectura de modelos persistentes, basado en los resultados del objetivo anterior.
- Implementar el *framework* basado en el diseño previamente realizado.
- Validar el *framework* a través de pruebas de aceptación [27].

3. Metodología

Al ser un proceso de desarrollo el objetivo de este trabajo de grado, la metodología a aplicar debe ser (bajo condiciones normales) una metodología de desarrollo de software, por esta razón se considera usar una adaptación del método *RUP* (*Rational Unified Process*) [14].

Este método de desarrollo consiste de cuatro etapas, cada una con su propio número de iteraciones y con criterios de satisfacción los cuales deben ser cumplidos antes de pasar a la siguiente etapa. Se busca lograr que el proceso sea adaptable, equilibrar las prioridades y mediante todos estos factores lograr elevar la calidad del producto final.

En la primera fase, la fase de concepción, buscó modelar adecuadamente los principales requisitos del sistema. La segunda fase, la fase de elaboración, busca delimitar la arquitectura base del sistema y refinar los requerimientos. En esta fase como resultado se cuenta con el listado de requerimientos de *ZoomTI++* junto con la priorización de los mismos, a su vez se cuenta con los documentos de especificación de requerimientos (*SRS*) y de arquitectura del sistema (*SAD*). La tercera fase, la fase de construcción se enfoca en la construcción de la mayoría del *software*, de igual manera es enfocada en varias iteraciones, de esta etapa surge el *framework* con las funcionalidades especificadas en la segunda fase.

II – MARCO TEÓRICO

Para comprender el desarrollo de *ZoomTI++* y el porqué de algunas de sus características propias es necesario conocer la teoría alrededor de la visualización de diagramas de software, del concepto y uso de las interfaces aumentables y del uso de dispositivos con reconocimiento de tecnología *multi-touch*.

El marco contextual muestra el mundo actual como se desarrollaba sin la existencia de *ZoomTI++*, haciendo uso de ejemplos concretos y se presentan percepciones de estos frente a lo que se propuso. El marco conceptual se encarga de presentar conceptos que son de alta importancia para la comprensión de lo expuesto en secciones posteriores.

1. Marco Contextual

La problemática expuesta en la sección anterior hace referencia a una oportunidad de mejora muy específica, el uso de diferentes tecnologías y conceptos para la visualización de modelos puede permitir a un usuario evitar cometer errores y aumentar su comprensión del modelo sobre el cual se encuentre trabajando.

Sin embargo sería una equivocación afirmar que es la primera vez que se intenta atacar dicha problemática, a continuación se presentan diferentes aproximaciones que han abordado esta situación de una forma u otra.

En primer lugar se presentan soluciones que hacen uso del concepto de *Zoomable User Interfaces*.

- *PAD++* [20]

Permite ayudar en la manipulación de objetos gráficos según el nivel de *zoom*, se puede considerar como uno de los creadores de las *Zoomable User Interfaces*. De igual manera plantea las bases sobre como modelar comportamiento importante para este tipo de interfaces, entre ellos pero no limitado a: Ubicación espacial, redistribución en pantalla, mostrar los niveles de detalle.

Es un hecho de alta importancia el resaltar que está diseñado para manejar con altos volúmenes de información.

En relación a *ZoomTI++* presenta algunas bases interesantes, en especial la posibilidad de manejar altos niveles de información, sin embargo carece de elementos importantes como la interacción con pantallas táctiles y la orientación hacia modelos de *software*.

- *Jazz: An Extensible Zoomable User Interface Graphics Toolkit In Java* [21]

Basada en *Java*, permite el desarrollo de gráficos con diferentes niveles de representación basado en el *zoom*. *Jazz* reúne elementos interesantes de variadas técnicas (visualización 2D, 3D), permite la creación de aplicaciones con control de *zoom*.

Es positivo que la herramienta haya recogido conocimientos en varios campos en un solo lugar, sin embargo su uso es complicado para cualquier tipo de usuario que no tenga suficiente experiencia con ella.

Se busca que *ZoomTI++* permita al usuario interactuar de una manera natural (sin necesidad de realizar un aprendizaje significativo para el uso del *framework)* y *Jazz* se encuentra lejos de incorporar este elemento de comodidad, además de no poseer un buen rendimiento cuando se alcanza un número alto de gráficos.

- *Designing a Digital Library for Young Children: An Intergenerational Partnership* [22]

Hace uso intensivo de *Jazz*, enfocado su desarrollo hacia los usuarios (niños), es importante resaltar que la parte visual es principal para el éxito de la aplicación, se mantiene comandos para la ejecución muy sencillos, evitando combinaciones no naturales para el usuario. Se siguió una metodología de desarrollo centrado en usuario para poder tomar esos detalles en consideración.

Lo positivo de este trabajo es el enfoque en el usuario y en facilitar la interacción del mismo con el programa, pese a ser usuarios muy diferentes a los que presenta esta propuesta es importante analizar las conclusiones que llegaron durante el proceso de desarrollo, todo hacia la disminución de riesgos. Lo negativo de esta aplicación radica en que el enfoque que usa para *ZUI* es demasiado sencillo y carece del grado de complejidad que si incluye *ZOOMTI++*.

- SHriMP Views [23]

Nació para cubrir la necesidad de poder observar altos niveles de información, se realizó a través de un método que respetara la estructura jerárquica de los datos y variando lo mostrado al usuario de acuerdo al nivel de *zoom* (*zoom* semántico).

Está enfocada en sus orígenes a la visualización de sistemas de *software*, es una aproximación similar al objetivo de este trabajo, por supuesto, sin incluir los componentes *multi-touch*.

Las soluciones ya presentadas permiten comprender hasta cierto grado el presente del uso de *Zoomable User Interfaces* para hacer mayor uso de la racionalidad del usuario a la hora de comprender información frente a sus ojos, las siguientes soluciones que se presentan se encuentran más orientadas hacia lo que se dirige *ZoomTI++*.

- *ZooMEnv* [1]

El trabajo del ingeniero Jaime Pavlich, "Un ambiente de Meta-Modelado y Visualización Basado en el paradigma de *Zoomable User Interfaces*" [1] presenta un acercamiento a las *Zoomable User Interfaces* y de las posibilidades que su uso abre para mejorar la productividad. Posteriormente se encarga de presentar una herramienta de modelamiento y meta-modelamiento que se encarga de facilitar la creación rápida de editores de modelado.

Este trabajo permite denotar la arquitectura de una aplicación basada en el paradigma *ZUI*, a un nivel de detalle alto, esta arquitectura plantea una base útil para la definición de herramientas enfocadas a resolver situaciones similares. De igual manera es de resaltar el énfasis del *paper*

hacia incentivar el trabajo futuro en el área. Fue construida en *Java* y usa la librería *Piccolo2D* (basado en *Jazz*).

- FV – GAIA [24]

El trabajo del ingeniero Federico Rodríguez se encuentra enfocado hacia la manera de visualizar y manipular la información contenida por grafos según el nivel de abstracción. La información contenida en herramientas *CASE* generalmente se encuentra organizada por grafos lo cual convirtió el trabajo del ingeniero Federico Rodríguez en un complemento bastante útil para la identificación de los requerimientos implementados por *ZoomTI++*.

Estos requerimientos a los que se hace referencia son extraídos a partir de la arquitectura definida por *ZooMEnv* (trabajo del ingeniero Pavlich [1]), de igual manera presenta una implementación de los requerimientos extraídos.

Eso es lo positivo, es una aproximación más cercana a lo que se buscaba con el desarrollo de este trabajo, sin embargo no incluye los elementos de tecnología *multi-touch* que son uno de los factores novedosos de esta propuesta.

- *Multitouch Navigation in Zoomable User Interfaces for Large Diagrams* [4]

Este trabajo es de los pocos que relacionan todos los elementos referentes a lo que buscaba *ZOOMTI*, es una aproximación desde dos perspectivas para visualizar información en grafos jerárquicos, una de estas aproximaciones usa un enfoque de tecnología *multi-touch*, una tabla táctil para ser más específicos, junto al *zoom* semántico característico de la navegación tradicional de dispositivos con tecnología *multi-touch*.

Lo interesante de esta aplicación es que permite conocer resultados preliminares de la interacción de tecnología *multi-touch* con *Zoomable User Interfaces*. Estos resultados son positivos al parecer, pero es necesario estudios más profundos para conocer a ciencia cierta el efecto sobre el usuario, esto debido a lo reciente del trabajo.

- *PyMT: A Post-WIMP Multi-Touch User Interface Toolkit* [25]

Creada sobre *Python* debido a la necesidad de una herramienta que se ajustara a las necesidades inmediatas de los desarrolladores para la creación de modelos tanto para usuarios avanzados como para usuarios novatos. A pesar de que no son muy notorios los elementos de *Zoomable User Interfaces* en esta aplicación, las bases sobre la cual se creó tuvieron en cuenta elementos basados en este paradigma. Es un buen ejemplo sobre las bases necesarias para tener en cuenta a la hora de realizar aplicaciones con interacción en tecnología táctil.

ZoomTI++ busca solucionar la problemática expuesta a partir del uso de tecnología *multi-touch* (para la navegación en los diagramas), de interfaces aumentables (para permitir mayor abstracción por parte del usuario) y de *C++* (para alcanzar altos niveles de rendimiento en diagramas con alto número de elementos). Las aproximaciones expuestas se acercan a esta solución en ciertos elementos pero ninguna incluye la mezcla exacta que si tiene *ZoomTI++*.

2. Marco Conceptual

Para garantizar una suficiente comprensión del *framework* desarrollado y de las explicaciones respecto al mismo es necesario poseer claridad sobre varios conceptos claves los cuales son presentados a continuación:

ZoomTI++ es un componente de un proyecto que busca desarrollar una herramienta *CASE*, la cual haga uso de los principios de *Model Driven Architecture*, la cual permite definir la estructura y comportamiento de un sistema utilizando lenguajes de modelamiento, para posteriormente ser transformados en código [1] y los principios de *Model Driven Engineering*, enfocada en la creación y uso de modelos de dominio.

El objetivo de toda herramienta *CASE* es automatizar métodos para mejorar la calidad del desarrollo y a partir de esto mejorar la calidad del producto final [28]. La herramienta definida en *ZoomEnv* [1] busca lograr esto al proveer un mecanismo de creación rápida de editores de grafos con sintaxis concretas basadas en *Zoomable User Interfaces*, dichas interfaces son caracterizadas por organizar la información en el espacio y permitir variar la información mostrada de acuerdo al nivel de *zoom* realizado por el usuario.[2] De manera paralela a dicho documento se definieron los requerimientos enfocados a la visualización a través de interfaces aumentables.

El trabajo de grado *FV-GAIA*: *Framework* para la Visualización de Grafos con Ayuda de Interfaces Aumentables [24], realizó una especificación formal de los requerimientos de dicha herramienta CASE, extendiendo los requerimientos presentados referentes a las interfaces aumentables y agregando los demás requerimientos que la herramienta final debería presentar en funcionamiento. De igual manera ese trabajo de grado realizó una implementación de los requerimientos detectados, en un *framework* denominado de la misma manera. *FV-GAIA* fue desarrollada en el lenguaje de programación JAVA [18] e incluye funcionalidades tanto para visualizar como para modificar los elementos existentes en los diagramas que especifican el *software* almacenados en forma de grafos. Un grafo es una representación compuesta de un número finito de nodos y conexiones (llamadas aristas) [27], definición que hace referencia a la estructura de datos.

Una aproximación que se encuentra actualmente en desarrollo durante el momento que la presente memoria fue escrita es el trabajo de grado de un estudiante de maestría, dicho trabajo busca implementar el componente visual de la herramienta *CASE* a partir de la extensión de requerimientos del trabajo de grado *FV-GAIA* [24], dicho trabajo responde al nombre *ZMT*. Una de las características de *ZMT* es que se está desarrollando en el lenguaje de programación HTML [19]. Sin embargo no se poseen mayores detalles por el momento.

Otra de las aproximaciones a la herramienta CASE definitiva es una nueva extensión de los requerimientos de *FV-GAIA* [24], cuya culminación es el presente trabajo de grado. Haciendo uso de los requerimientos definidos en *FV-GAIA* [24], se planteó el módulo de visualización de la herramienta *CASE* [28], sin embargo, se realizaron las labores de ingeniería de requerimientos para encontrar los requerimientos asociados al uso de tecnología *multi-touch*.

Se buscó crear un *framework* para visualizar diagramas de *software*; estos son una representación gráfica que busca mejorar la comprensión del funcionamiento de un modelo, los cuales a su vez son el resultado de abstraer propiedades relevantes de entidades del mundo real. [26] [29]

ZoomTI++, desarrollado en el lenguaje de programación C++ (el cual tiene la posibilidad de realizar sobre él programación imperativa, orientada a objetos o genérica [15]), brinda al usuario una interfaz gráfica, con la que le permite pasar información al sistema [27]. La información es enviada a través del uso de tecnología *multi-touch*, lo que permite hacer uso de las manos y eliminar la necesidad de usar un dispositivo adicional.

Haciendo uso de esta tecnología y del *zoom* semántico, el cual permite variar la representación de diferente información (tanto en cantidad, tipo de información, organización de la misma) según el nivel de *zoom* sostenido en determinado punto [2], el usuario puede explorar el diagrama haciendo uso de razonamiento espacial.

De esta manera se buscó acercarse a la oportunidad de mejora mencionada en las secciones anteriores de la presente memoria.

A continuación se presenta en detalle la manera en la que este *framework* se volvió realidad.

III – CONCEPCIÓN

En la primera fase, la fase de inicio, se buscó modelar adecuadamente los principales requisitos del sistema dentro de los que se encuentra:

• Estado del arte: Se desarrolló en el tercer periodo del año 2014, en donde están especificados los temas necesarios para el desarrollo de *ZoomTI++*.

• Se seleccionaron los aspectos clave que debía incluir la aplicación, es decir, se definió a gran escala que elementos eran imprescindibles para empezar con el desarrollo, adaptándose inicialmente a los del trabajo de grado *FVGAIA* [24].

• Al saber que aspectos ya se tienen, se agregaron los aspectos novedosos de *multi-touch* y desarrollo bajo el lenguaje de programación *C++*.

• Se validó cada uno de estos aspectos con el cliente y se recibió su posterior aprobación.

Al terminar con estos requisitos de sistema, se dio por terminada esta iteración y se avanzó a la segunda fase, elaboración.

IV – ELABORACIÓN

Los procesos descritos en esta sección hacen referencia a la segunda fase de la metodología expuesta previamente.

1. Especificación de requerimientos

Los requerimientos que se presentan son el resultado de consideraciones acerca del tipo de usuario, el tipo de acciones que dichos usuarios deberían poder realizar de una manera rápida y sencilla, así como de restricciones determinadas durante la fase de concepción de *ZoomTI++*.

1.1 En relación al framework

Se espera que *ZoomTI++* haga parte de una futura herramienta *CASE*, definiendo el componente de visualización de diagramas; la herramienta final sería capaz de no solo visualizar, sino también de crear y guardar diagramas, por mencionar un par de sus potenciales características. Por esta razón *ZoomTI++* se encuentra directamente enfocado a dos campos: Concentrarse en visualización (desde la carga de la especificación visual del modelo, hasta verlo en la pantalla) y en permitir el escalamiento del sistema; al ser un componente de una posible herramienta CASE, se busca que un posible usuario futuro pueda agregar funcionalidades (soporte de más primitivas, distintos *layouts*, formatos adicionales para la lectura de diagramas) sin que *ZoomTI++* sea una limitante para esto, sino por lo contrario, *ZoomTI++* permita hacer uso de sus bondades con estos nuevos elementos sin necesidad de modificar componentes ya existentes.

Para lograr dicho objetivo el *framework* es capaz de:

- Leer la información del diagrama, el cual representa de forma visual al modelo.
- Almacenar los datos de dichos diagramas en primitivas que almacenen la totalidad de la información.
- Visualizar en pantalla a partir de dichas primitivas, el diagrama.
- Permitir el uso de gestos *multi-touch* para navegar en el diagrama; *tap*, *double tap*, *drag*, *pinch*, *spread*, *press*, entre otros. [30]
- Variar la información mostrada en pantalla según el nivel de *zoom*; *zoom* semántico.
- Organización de la información según *layouts* específicos

ZoomTI++ brinda una manera de resolver dichas necesidades, sin embargo permite que un desarrollador sea capaz de agregar sistemas de lecturas de diagramas, siempre que sigan la estructura definida por el *framework*. De igual manera se brinda la posibilidad de agregar primitivas, más *layouts*, etc. *ZoomTI++* se encuentra orientado hacia su continuo crecimiento.

Una herramienta *CASE* completa debería ser capaz de administrar modelos, es decir debería tener mínimo las siguientes funcionalidades: crear modelos, editar modelos, eliminar modelos y visualizar modelos. Esta es una manera general de enunciar y agrupar lo que se esperaría de una herramienta de este tipo. Como ya se indicó, el objetivo de *ZoomTI++* está dirigido hacia la visualización de diagramas de *software* (la cual es una forma de representar visualmente el modelo). Los demás elementos que hacen parte de la administración de modelos, quedan fuera del alcance de *ZoomTI++*.

1.2 En relación al usuario

Un desarrollador que busque hacer uso de *ZoomTI++* para fines de tomarlo como componente de una herramienta *CASE*, debe estar en capacidad de realizar la integración con otros sectores de la herramienta. Debe ser capaz de comprender las estructuras definidas del sistema y por lo tanto implementar nuevas funcionalidades que se incorporen de una manera funcional al *framework.*

Desde el punto de vista de un usuario de *ZoomTI++*, se debe estar en capacidad de comprender el formato de entrada ya definido y de esta manera ser capaz de crear diagramas estructuralmente correctos para ser visualizados en el *framework*.

1.3 En relación a las condiciones del *framework*

Pese a ser una aproximación nueva en muchos aspectos, es de igual manera cierto el hecho de que no es la primera vez que se realiza un *framework* con las características ya descritas, por esta razón se tuvo en cuenta algunas dependencias y algunas restricciones para la elaboración de *ZoomTI++*.

En el contexto universitario sobre el cual fue planteado originalmente *ZoomTI++*, ya existían aproximaciones similares realizadas en Java [18] y en HTML5 [19]. Por esta razón se ve como una restricción que la elaboración de *ZoomTI++* fuera usando un lenguaje de programación diferente a estos, en este caso, C++ esto es positivo, debido a que refuerza otra de las dependencias a considerar, el rendimiento del *framework.*

Otra restricción radicó en el dispositivo con soporte *multi-touch* a disposición, los drivers del mismo presentan un funcionamiento impecable sobre sistemas operativos *Windows*, razón por la cual *ZoomTI++* fue elaborado tomando esto en consideración.

1.4 Listado de requerimientos

Fueron identificados 39 requerimientos los cuales son considerados suficientes para hacer frente a las necesidades expuestas [41].

Código del requerimiento	Requerimiento	Tipo
REQ 001	El framework debe permitir realizar la lectura de un archivo con formato .JSON, el cual haga uso de la especificación expuesta en el manual de plataforma.	Funcional
REQ 002	El framework debe permitir visualizar en pantalla el diagrama cargado previamente.	Funcional
REQ 003	El framework debe permitir realizar Zoom In en el diagrama, a través de la interacción con la pantalla Multi-Touch.	Funcional
REQ 004	El framework debe permitir realizar Zoom Out en el diagrama, a través de la interacción con la pantalla Multi-Touch.	Funcional

REQ 005	El framework debe permitir desplazamiento (eje vertical y eje horizontal) en el diagrama, a través de la interacción con la pantalla Multi-Touch.	Funcional
REQ 006	El framework debe permitir al programador crear nodos y aristas por medio de primitivas gráficas en forma vectorial (líneas, rectángulos, óvalos, imágenes y texto).	Funcional
REQ 007	El framework debe permitir al programador crear y aplicar transformaciones en elementos básicos de grafos (nodos y aristas).	Funcional
REQ 008	El framework debe manejar nodos que tengan la propiedad de anclas.	Funcional
REQ 009	El framework debe manejar aristas y nodos contenedores.	Funcional
REQ 010	El framework debe manejar nodos que tengan la propiedad de realizar grupos de nodos.	Funcional
REQ 011	El framework debe manejar transformaciones geométricas aplicadas a nodos y aristas.	Funcional
REQ 012	El framework debe manejar transformaciones geométricas teniendo en cuenta como la transformación puede aplicar a un nodo o arista contenedor.	Funcional
REQ 013	El framework debe manejar transformaciones geométricas teniendo en cuenta como la transformación puede aplicar a nodos que tienen la propiedad de anclas.	Funcional
REQ 014	El framework debe manejar transformaciones geométricas teniendo en cuenta como la transformación puede aplicar si es un grupo de nodos.	Funcional
REQ 015	El framework debe guardar la geometría de primitivas gráficas y los datos expuestos en el diagrama.	Funcional
REQ 016	El framework debe permitir visualizar información distinta del diagrama, dependiendo del nivel de zoom algunos elementos deben mostrarse u ocultarse.	Funcional
REQ 017	El framework debe cambiar la distribución en pantalla de los elementos del diagrama, de acuerdo al nivel de zoom.	Funcional
REQ 018	El framework debe organizar los elementos de grafos con layouts circulares.	Funcional
REQ 019	El framework debe organizar los elementos de grafos con layouts jerárquicos.	Funcional
REQ 020	El framework debe organizar los elementos de grafos con layouts ortogonales.	Funcional
REQ 021	El framework debe organizar los elementos de grafos con layouts en árbol.	Funcional
REQ 022	El framework debe organizar nodos que pertenezcan a un contenedor o a un grupo de nodos por medio del layout matricial.	Funcional
REQ 023	El framework debe organizar nodos que pertenezcan a un contenedor o a un grupo de nodos por medio del layout padre (con respecto a la posición medida desde el origen del nodo padre).	Funcional

REQ 024	El framework debe organizar los nodos contenidos dentro de una arista con layout de coordenadas (que los hijos se acomoden por medio de coordenadas y distancia de una arista).	Funcional
REQ 025	El framework debe organizar los nodos contenidos dentro de aristas con layout de posicionamiento (Los hijos de las aristas se acomoden en el espacio de forma que no se oculten unos objetos con otros).	Funcional
REQ 026	El framework, al alcanzar un nivel de zoom alto sobre un nodo en específico, debe manejar el layout de nodos adyacentes (reposicionar los nodos con que se relaciona directamente el nodo enfocado).	Funcional
REQ 027	El framework debe permitir cambiar la representación visual de elementos dependiendo del nivel de zoom.	Funcional
REQ 028	El framework debe alcanzar un desempeño que logre pasar las pruebas especificadas en el documento "Selección Plataforma".	No Funcional
REQ 029	El framework debe informar a los usuarios una descripción de los pasos requeridos para realizar cada uno de las funcionalidades que ofrece el sistema, por medio de un manual de programador.	No Funcional
REQ 030	El framework debe estar programado en C++.	No Funcional
REQ 031	El framework debe soportar ejecución en sistemas operativos Windows.	No Funcional
REQ 032	El framework debe soportar el uso de tecnología Multi-Touch.	No Funcional
REQ 033	El framework debe soportar imágenes en formato vectorial.	No Funcional
REQ 034	El framework debe permitir hacer Zoom In en un nodo específico, realizando doble clic en la pantalla Multi-Touch.	Funcional
REQ 035	El framework debe permitir hacer Zoom Out de un nodo específico, realizando doble clic en la pantalla Multi-Touch.	Funcional
REQ 036	El framework debe permitir volver al nivel de Zoom inicial, a través del uso de un botón.	Funcional
REQ 037	El framework debe permitir volver al nivel de Zoom del nodo padre, a través del uso de un botón.	Funcional
REQ 038	El framework debe permitir realizar la lectura de un nuevo archivo con formato .JSON, a través del uso de un botón.	Funcional
REQ 039	El framework debe organizar las aristas que unen dos nodos por medio de un layout de aristas (reubicar las aristas de forma que una arista no se oculte con otras).	Funcional

Tabla 1 Listado de requerimientos identificados.

1.5 Priorización de requerimientos

La priorización de requerimientos es un paso esencial para orientar el desarrollo de un proyecto de *software* de manera que los requerimientos implementados sean los de mayor impacto para el criterio usado. La priorización de *ZoomTI++* siguió el criterio denominado: costo y riesgo [31].

En dónde para cada requerimiento se analizaron los siguientes ítems [24]:

- **Beneficio relativo:** El beneficio se refiere como tal a la perspectiva del negocio del producto de software, es decir, el requerimiento al ser implementado qué grado de beneficio trae al ser implementarlo.
- **Pena relativa:** Define que tan fácil (1) o difícil (10) será la implementación.
- **Valor total:** La suma del beneficio relativo (multiplicado por dos en este caso por ser de gran importancia para el trabajo de grado) y la pena relativa.
- **Valor %:** El porcentaje del valor con respecto al total.
- **Costo relativo:** Este cálculo se basa en la complejidad del requerimiento, la extensión de la interfaz de usuario de trabajo, la reutilización de diseños existentes, niveles de prueba y de documentación.
- **Costo %:** El porcentaje del costo con respecto al costo total.
- **Riesgo relativo:** Se refiere a la dificultad a la hora de programar.
- **Riesgo %:** El porcentaje del riesgo con respecto al riesgo total.
- **Prioridad:** Se define mediante la fórmula, esto teniendo en cuenta el peso en el valor del beneficio relativo aclarado previamente:
    - Valor% / (Riesgo % + Costo%)

A partir de los resultados obtenidos se pasó a conversar con el cliente para evaluar lo obtenido. Seis requerimientos puntuales, después de conversar con el cliente, fueron considerados fuera del alcance del proyecto. Esto debido a que el cliente manifestó que dichos requerimientos son de poca importancia frente a sus expectativas puntuales con *ZoomTI++*, sin embargo se dejan planteados como posible trabajo futuro. [41]

REQ 019	El framework debe organizar los elementos de grafos con layouts jerárquicos.
REQ 020	El framework debe organizar los elementos de grafos con layouts ortogonales.
REQ 021	El framework debe organizar los elementos de grafos con layouts en árbol.

REQ 025	El framework debe organizar los nodos contenidos dentro de aristas con layout de posicionamiento (Los hijos de las aristas se acomoden en el espacio de forma que no se oculten unos objetos con otros).
REQ 026	El framework, al alcanzar un nivel de zoom alto sobre un nodo en específico, debe manejar el layout de nodos adyacentes (reposicionar los nodos con que se relaciona directamente el nodo enfocado).
REQ 039	El framework debe organizar las aristas que unen dos nodos por medio de un layout de aristas (reubicar las aristas de forma que una arista no se oculte con otras).

Tabla 2 Listado de requerimientos descartados durante la priorización

1.6 Licenciamiento

ZoomTI++ fue desarrollado usando QT, que cuenta con licencia LGPL v3. Este tipo de licencia (*Lesser General Public Licens*e) es licencia de uso abierto, la cual es recomendable que sea utilizada también por cualquier tipo de aplicación que haga uso de dicha librería [32]. Por esta razón *ZoomTI++* es un *framework open source* con licencia LGPL v3 [42].

2. Selección de la herramienta

En cuanto al diseño de *ZoomTI++* se tuvo como primera etapa la selección de la herramienta, que se llevó a cabo de la siguiente manera y con estos candidatos:

- **QT [33]:** Herramienta implementada en *C++* diseñada para desarrollo de *software* multiplataforma de una manera sencilla. Cuenta con una base de usuarios cercana a medio millón de desarrolladores.
  - Soporta desarrollo en *C++* y *QML*.
  - Presenta compatibilidad con sistemas operativos: *Windows*, *Linux*, *OSX*.
  - Es enfocado a la creación de aplicaciones e interfaces de usuario.
  - Cuenta con librerías y ejemplos para el uso de tecnología *multi-touch*.

- **openFrameworks [34]:** Conjunto de librerías multiplataforma con funciones de propósito general mediante el uso del lenguaje de programación C++. Cuenta con las funcionalidades incluidas de otros *framework* como:

  - OpenGL, GLEW, GLUT, libtess2 para gráficas (que es de nuestro interés).
  - rtAudio, PortAudio, OpenAL y Kiss FFT oFMOD para entradas y salidas de audio.
  - FreeImage para guardar y cargar imágenes.
  - Quicktime, GStreamer y videoInput para reproducción de video.
  - Poco, para variedad de utilidades
  - OpenCV para visión de computador.

Se puede agregar a su vez librerías para el reconocimiento de la tecnología *touch*.

- **Eagle Mode [35]:** Es una aproximación a una interfaz 100% bajo el paradigma de las interfaces aumentables (ZUI). Es *open source* y cuenta con un API en C++ para desarrolladores.

    - Presenta compatibilidad con sistemas operativos: *Windows*, *Linux*.
    - La *API* permite desarrollo en *C++* pero *Eagle Mode* requiere ejecución bajo Perl.
    - A pesar de contar con una versión para *Windows*, esta carece de un largo número de funcionalidades respecto a la versión para sistemas operativos *Linux* (zonas horarias, reproductor de audio y video, visor de PDF, visor de SVG).
    - Cuenta con una pequeña cantidad de ejemplos de desarrollo.

- **GTK [36]:** Es un conjunto de librerías multiplataforma para desarrollar *GUI* de uso genérico y cuenta con las siguientes librerías:

    - GLib.
    - GTK.
    - GDK.
    - ATK.
    - Pango.
    - Cairo.

- **TouchLib [37]:** Librería *Open Source* escrita en *C++* y *OpenCV*. Permite llevar un registro de las señales obtenidas usando tecnología *multi-touch* (dedo abajo, dedo arriba, movimiento, etc).

    - La última versión fue liberada en el 2007.
    - Carece de soporte o de una comunidad activa, parece un proyecto abandonado.

- **Microsoft Visual Studio / Windows.h [38]:** Ambiente integrado de desarrollo de *Microsoft*. Idealmente permite el desarrollo orientado a aplicaciones a ejecutarse sobre sistema operativo *Windows*, aplicaciones web, páginas web y servicios web.

    - Windows.h es una librería que incluye funciones para la recepción, registro y manejo de eventos táctiles.
    - Permite desarrollo en: C, C++, C#, Visual Basic y F#.
    - Documentación completa en el sitio web para desarrolladores de Microsoft.

Detalles Estudio

Herramienta	Facilidad de instalación	Usabilidad	Licencia	Documentación	Plataforma
QT	Es por medio de un asistente, que instala todas las librerías necesarias y deja la herramienta lista para su uso. No requiere más de 5 minutos.	Hacer evento del zoom por medio de la rueda del mouse no toma más de 10 minutos, que hace tanto Zoom In como Zoom Out, y es algo muy parecido a lo que busca ZoomTI++. Los ejemplos que tiene QT son fáciles de abrir y entender, pues se encuentran disponibles en la interfaz incluidos en la instalación.	GPL v3 LGPL v3 LGPL v2	Cuenta con toda la documentación necesaria en la página oficial	Multiplataforma
openFrameworks	Se descarga un conjunto de librerías que se agregan manualmente a ya sea CodeBlocks o VisualStudio, en este caso se agregaron a CodeBlocks. Aunque no es trivial la instalación es un proceso bien explicado que lleva no más de 10 minutos.	Abrir los ejemplos que se descargan de forma gratuita de la página, no toma más de 5 minutos, y se entienden de forma sencilla	MIT License.	Cuenta con toda la documentación necesaria en la página oficial	Multiplataforma
Eagle Mode	Se descarga el paquete source, el cual contiene todas las clases de la librería. Dicho paquete debe ser instalado, para la instalación es necesario contar con Perl en el sistema operativo, así como GCC (esto	Los ejemplos deben ser ejecutados mediante Perl en línea de comandos. Es fácil una vez se realiza una primera vez.	GPL v3	La totalidad de la API se encuentra documentada, sin embargo la documentación carece de información detallada.	Multiplataforma

	implica más labores de instalación). Posteriormente ejecutar un par de líneas en consola. Para una instalación que requiera instalar los elementos enunciados se requiere alrededor de 45 minutos.				
GTK	Al descargar el paquete requiere ser descomprimido y añadir dicha librería a las variables del sistema. A su vez se necesita configurar gcc en las variables del sistema, para poder compilar desde la misma consola. El tiempo aproximado, pues se necesitan saber algunos comandos es de 30 minutos.	Los ejemplos se ejecutan por línea de comando. Son sencillos y se puede ver el código fuente	LGPL	Cuenta con toda la documentación necesaria en la página oficial	Multiplataforma
Touch-Lib	Para el modo desarrollador se requiere compilar la solución del proyecto en Visual Studio	Los ejemplos incluidos en la librería no son fáciles de usar, sin embargo cuenta con herramientas para la calibración de los dispositivos.	BSD 2	Muy poca documentación disponible.	Windows
Microsoft Visual Studio / Windows.h	Para hacer un uso completo de las capacidades de la librería es necesario contar con una instalación de Visual Studio en el equipo. Instalación rápida a través del asistente.	Los ejemplos disponibles en MSDN son muy específicos y en teoría deben funcionar. Sin embargo la creación de los proyectos	Al ser software propietario de Microsoft la licencia se encuentra condicionada, sin embargo la edición express permite	MSDN cuenta con documentación bastante completa, de igual manera hay a dispo-	Windows

		específicos y la realización de labores puntuales presentan una dificultad alta para usuarios inexpertos en el uso de Visual Studio.	distribuir sin ninguna restricción particular.	sición ejemplos puntuales en C++.	

Tabla 3 Detalles estudio herramienta.

En la siguiente tabla se muestra

- Facilidad de instalación: Percepciones acerca de la dificultad para realizar la instalación de dicho elemento.
- Usabilidad: Percepciones acerca de la complejidad para cargar un elemento de prueba y ejecutarlo.
- Licencia: Se especifica el tipo de licenciamiento con el cual queda vinculada una aplicación desarrollada haciendo uso de la herramienta. Se considera apto todo tipo de licenciamiento libre para la aplicación. Entre menos restrictivo es el tipo de licenciamiento usado, mayor puntaje se le asigna (1 como máximo) y entre más restricciones presenta la licencia este valor disminuye hasta un valor mínimo de 0, a continuación se describe con algunos ejemplos los valores de cada tipo de licencia.
    - Licencias como "MIT License", "Apache" y "BSD" tienen valor de 1.
    - Licencias como "LGPL", "GPL" y la licencia relacionada a aplicaciones realizadas en "Visual Studio" tienen valor de 0.5.
    - Licencias privadas tienen valor de 0.
- Documentación: Percepciones acerca de la facilidad para encontrar contenido referente al uso de la herramienta.
- Plataforma: Se especifica si la herramienta funciona sobre una plataforma específica o si al contrario es multiplataforma.

La siguiente tabla Asigna puntuaciones a los criterios expuestos en la Tabla anterior. Dichas puntuaciones oscilan entre 0 y 1, basado en la manera de calificar expuesta en la explicación de la tabla anterior.

Herramienta	Facilidad de Instalación	Usabilidad	Licencia	Documentación	Plataforma	**Total**
QT	1	1	0.5	1	1	**4.5**
Open Frameworks	0.7	0.7	1	1	1	**4.4**
Eagle Mode	0.2	0.5	0.5	0.5	1	**2.7**
GTK	0.4	1	0.5	1	1	**3.9**
TouchLib	0.2	0.5	1	0.25	1	**2.95**

Visual Studio/Windows.h	1	0.5	0.5	1	1	**4**

Tabla 4 Puntuaciones estudio herramientas

La siguiente tabla muestra los resultados totales de cada una de las herramientas según los criterios ya establecidos:

Herramienta	Total
QT	69,5
Open Frameworks	69,2
Eagle Mode	52,7
GTK	55,7
TouchLib	33,75
Visual Studio/Windows.h	68

Tabla 5 Puntuaciones finales preselección

Como se puede observar, las tres herramientas con mayor puntaje fueron en orden: QT, Visual Studio y Open Frameworks. Sobre estas fueron aplicadas las pruebas para encontrar a la mejor candidata para *ZoomTI++*.

Se procedió a aplicar pruebas a las tres herramientas finalistas: QT, Visual Studio y Open Frameworks. El objetivo de dichas pruebas era evaluar que herramienta funciona mejor bajo situaciones de exigencia alta, similares a las esperadas con *ZoomTI++* cuando se encuentre en su estado final.

A pesar de ser finalista, Visual Studio tuvo un puntaje muy bajo en el criterio de usabilidad en relación a las otras dos. No fue labor sencilla la creación de una aplicación bajo las condiciones requeridas para *ZoomTI++*. A pesar de que en los demás criterios: viabilidad con los requerimientos, licenciamiento, etc. obtuvo puntajes altos, este elemento de usabilidad es suficiente motivo para descartar a Visual Studio del grupo final de herramientas candidatas, cuando se toma en contraste los puntajes altos obtenidos en usabilidad por parte de las otras herramientas finalistas esta decisión se ve reforzada.

Dos de los factores fundamentales de *ZoomTI++* es la parte visual y la parte de actualización de la interfaz gráfica. A partir de estos hechos se diseñó un sistema de pruebas que se encargaran de exigir a cada uno de los finalistas en estos campos. De igual manera se evaluaron los siguientes aspectos en las herramientas:

- Pruebas de carga del procesador y memoria: Se evalúa la exigencia sobre el equipo al ser sometido a las situaciones creadas las pruebas. Se evalúa tanto uso de CPU (porcentaje de uso del procesador en todos los núcleos) como de memoria RAM (Memoria física total reservada).

- Pruebas de stress: Se evalúa la resistencia de la herramienta a las situaciones de las pruebas, tratando de determinar si el sistema llega a un punto donde deja de funcionar correctamente.

- Pruebas de tiempo de respuesta: Buscan cuantificar con tiempos acerca de la velocidad con la que las herramientas procesan las solicitudes, esto permite obtener un dato puntual acerca del rendimiento de las mismas.

Siguiendo el formato de prueba presentado a continuación:

Pruebas de Desempeño					
No.Prueba:					
Fecha:					
Sistema Operativo:					
Caracteristicas del Equipo:					
Nombre del componente:					
Protocolo de pruebas:					
Pruebas:	**Aspectos**	**Que se probó?**	**Prueba 1**	**Que se probó?**	**Prueba 2**
TIPOS DE PRUEBAS DE DESEMPEÑO					
Pruebas de carga del procesador y memoria	*Uso intensivo de la CPU*	Dibujar 10000 líneas		Dibujar 100000 líneas	
		Dibujar 20000 líneas		Dibujar 200000 líneas	
		Dibujar 10000 líneas, limpiar pantalla y repetir 10 veces		Dibujar 100000 líneas, limpiar pantalla y repetir 10 veces	
		Dibujar 20000 líneas, limpiar pantalla y repetir 20 veces		Dibujar 200000 líneas, limpiar pantalla y repetir 20 veces	
	Uso intensivo de la memoria	Dibujar 10000 líneas		Dibujar 100000 líneas	
		Dibujar 20000 líneas		Dibujar 200000 líneas	
		Dibujar 10000 líneas, limpiar pantalla y repetir 10 veces		Dibujar 100000 líneas, limpiar pantalla y repetir 10 veces	
		Dibujar 20000 líneas, limpiar pantalla y repetir 20 veces		Dibujar 200000 líneas, limpiar pantalla y repetir 20 veces	
Pruebas de Stress	*Limite*	Dibujar 10000 líneas		Dibujar 100000 líneas	
		Dibujar 20000 líneas		Dibujar 200000 líneas	
		Dibujar 10000 líneas, limpiar pantalla y repetir 10 veces		Dibujar 100000 líneas, limpiar pantalla y repetir 10 veces	
		Dibujar 20000 líneas, limpiar pantalla y repetir 20 veces		Dibujar 200000 líneas, limpiar pantalla y repetir 20 veces	
uebas de tiempo de respues	*Tiempo de Respuesta*	Dibujar 10000 líneas		Dibujar 100000 líneas	
		Dibujar 20000 líneas		Dibujar 200000 líneas	
		Dibujar 10000 líneas, limpiar pantalla y repetir 10 veces		Dibujar 100000 líneas, limpiar pantalla y repetir 10 veces	
		Dibujar 20000 líneas, limpiar pantalla y repetir 20 veces		Dibujar 200000 líneas, limpiar pantalla y repetir 20 veces	

Tabla 6 Formato Prueba Herramienta

Para QT las pruebas arrojaron resultados positivos en las pruebas de carga del procesador y memoria, en las pruebas de *stress* se obtuvieron valores bajos de consumo y la herramienta respondió a las solicitudes. Al tratar de dibujar un alto número de líneas el tiempo obtenido en

las pruebas de tiempo de respuesta fue bastante corto; de 10000 líneas a 100000 líneas hubo un aumento en tiempo en cantidades esperadas, nunca desproporcionadas.

En cuanto a Open Frameworks, las pruebas fueron de igual manera positivas, aprobó sin ningún problema las pruebas de carga del procesador y memoria así como las pruebas de stress, adicionalmente las pruebas de tiempo de respuesta obtuvieron tiempos bastantes buenos, en parte se puede deber a la manera en la que el conjunto de librerías ya trae implementación de la recarga de pantalla.

Las pruebas fueron realizadas en dos equipos con especificaciones diferentes, siendo la diferencia más representativa que uno de los equipos posee una tarjeta de video dedicada (Nvidia de 2gb) mientras que el otro posee solamente una integrada (intel de 1 gb). Al observar las diferencias entre un equipo y otro con el mismo escenario se encuentra que efectivamente un equipo usando una tarjeta de video dedicada *ZoomTI++* - Selección Plataforma 11 arroja resultados más positivos en todas las pruebas; menor uso de CPU, menor uso de memoria RAM, tiempos de respuesta más cortos.

Como se puede observar, ambas herramientas obtuvieron resultados similares y eran candidatas aptas para ser la base sobre la cual *ZoomTI++* se desarrollaría. Para lograr alcanzar una decisión final se enfocó concretamente en encontrar una librería compatible con cada herramienta para la interacción con pantalla *multi-touch,* dicha investigación se tuvo en cuenta en el análisis previo de herramientas.

El problema con algunas de las librerías consultadas en OpenFrameworks es que fueron desarrolladas hace tiempo, ejemplo concreto, la librería ofxBlackBox cuya última versión fue cargada a GitHub en el 2011 [51], esto generó que al usar otras librerías actualizadas, como ofxTUIO (librería esencial para el funcionamiento de ofxBlackBox) se generarán errores al hacer uso de métodos cuyos nombres cambiaron, esto generó la necesidad de realizar una actualización manual del código de ofxBlackBox solo para poder ejecutar un ejemplo. Esta clase de problemas pueden repetirse debido al concepto sobre el cual gira OpenFrameworks, librerías independientes que trabajan en conjunto.

En cambio con QT, los ejemplos vienen incluidos en el paquete de instalación y se encuentran listos para usarse, esto permite ejecución directa que facilita el desarrollo. A partir de todo el proceso y tomando en consideración hasta el último aspecto, se decidió usar como herramienta a QT.

2.1 Configuración del entorno

Después de definir el uso de Qt como base, se pasó a configurar todos los elementos necesarios para que el comienzo de la siguiente fase (construcción) no se viera afectado. Para ese momento ya se tenía a disposición un espacio en la sala de investigación SIDRe, en dicho espacio se contaba con un equipo, el cual contaba con sistema operativo *Windows* (uno de los requerimientos) y se tenía a disposición una pantalla con soporte a tecnología *multi-touch.*

Sobre dicho equipo se pasó a configurar lo necesario para el proceso de desarrollo; Instalación de *QT Creator* (para el manejo del proyecto) y de *Git* (para manejar el repositorio de código).

Durante el proceso de selección de herramienta se realizaron tutoriales de cada herramienta, por supuesto incluyendo *QT Creator*, razón por la cual ya se poseía conocimiento acerca de su uso básico. El único paso restante era aprender a usar *Git* (hacer *request* de *pull*, *commit*, *add*, *remove*, etc.) y con esto darle un correcto manejo al repositorio.

3. Diseño del sistema

3.1 Vista lógica

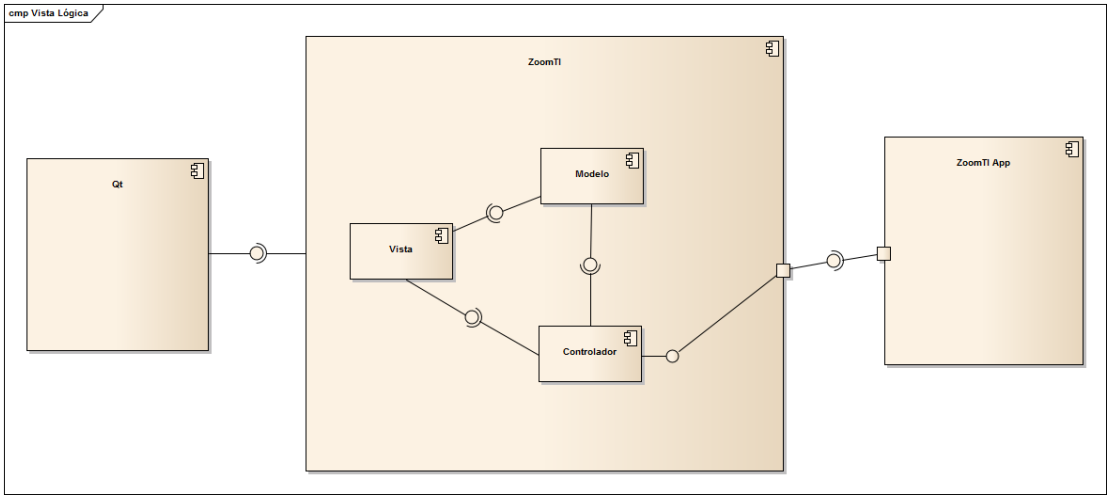


Ilustración 1 Arquitectura de alto nivel ZoomTI

El componente *Qt* [43] que cuenta con las librerías necesarias para facilitar la interfaz gráfica, provee su interfaz que es usada *ZoomTI*, el cual contiene 3 componentes básicos que son el *Modelo*, *Vista* y el *Controlador*.

El *Controlador* y la *Vista* acceden a la interfaz del modelo, el primero para saber cómo se componen los elementos gráficos, y de este modo darle comportamiento los mismos, la *Vista* por otra parte, necesita de esta interfaz para saber qué dibujar y en donde dibujarlo.

La *Vista* provee su interfaz al Controlador, para que este pueda saber que comportamientos pueden ser modificados en la vista, y de qué modo se pueden hacer dichos cambios. Al saberlo junto con la interfaz del Modelo, el controlador tiene todo lo necesario para controlar la escena de *ZoomTI*.

Por último al crear la aplicación se genera el paquete *ZoomTI App*, que referencia al prototipo en funcionamiento, que accede a la interfaz dada por *ZoomTI.*

3.2 Vista de despliegue

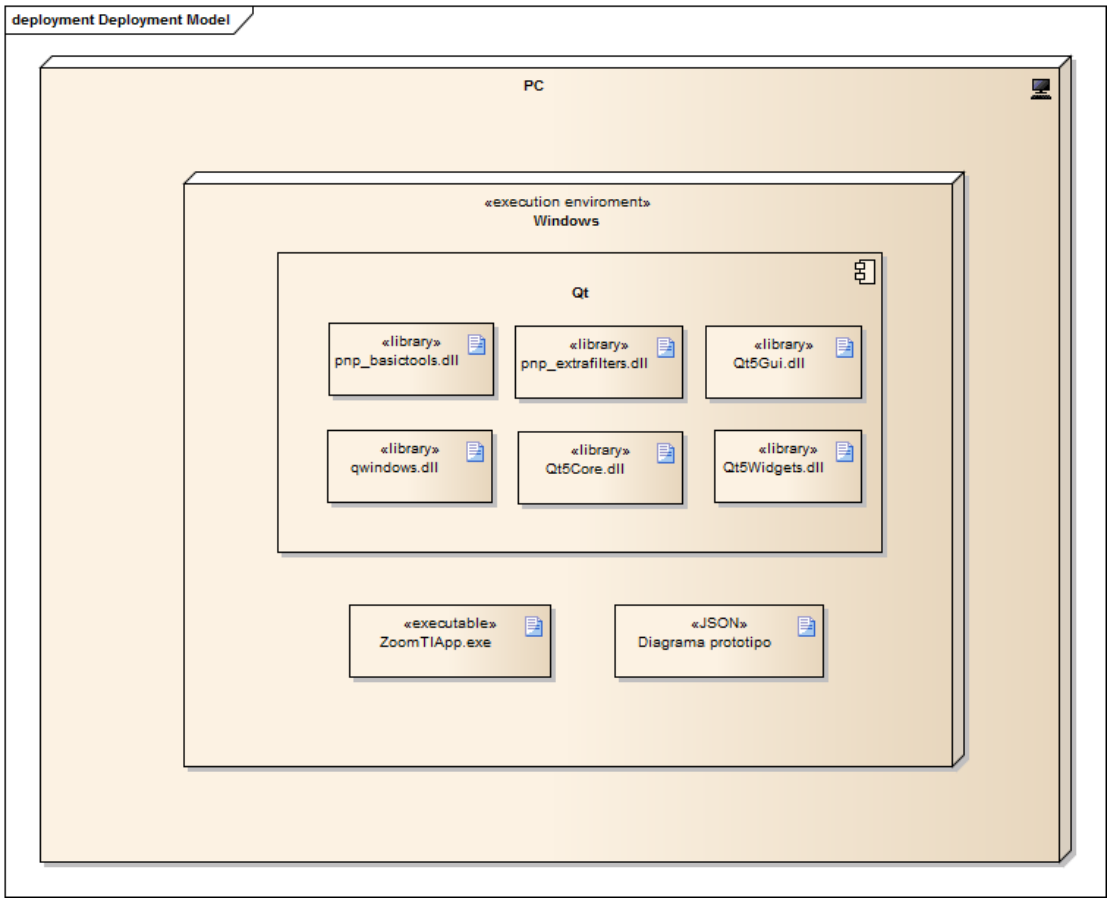


Ilustración 2 Despliegue ZoomTI

Se cuenta dos nodos, *PC* (primer nodo), que es el computador con sistema operativo *Windows* (segundo nodo) en donde está alojado el componente *Qt* [43]. Para que el componente *Qt* pueda funcionar es necesario tener las librerías contenidas dentro del mismo mostradas en la *Ilustración 2*. El ejecutable del *framework* se denota como *ZoomTIApp.exe*, generado por el componente *Qt*, este ejecutable a su vez es el responsable de acceder a una especificación de diagrama de software en formato *JSON* que está alojado en el primero nodo, todo esto para su correcto funcionamiento.

V – CONSTRUCCIÓN

Al momento de empezar a desarrollar el *framework* se tenía un estado avanzado en la parte investigativa, de planeación y de diseño: Se tenía definida a *QT* como base para el desarrollo, se había definido la arquitectura del *framework* y se contaba con un listado de requerimientos a implementar con su respectiva priorización para determinar el orden de la implementación.

Previamente se definió la metodología como *RUP*, pensando en realizar iteraciones cortas durante el transcurso del desarrollo. En este punto se alcanzó la tercera fase de la metodología expuesta al comienzo de esta memoria, habiendo aprobado las dos primeras etapas. A continuación se profundiza sobre la tercera fase (desarrollo).

Se planteó el uso de iteraciones durante esta etapa para lograr recibir realimentación del cliente en forma continua y poder evaluar la aceptación del mismo frente a cada requerimiento implementado. La manera en la que *ZoomTI++* fue diseñada requería implementar funcionalidades puntuales y luego evaluar su integración con componentes ya existentes. Para poder lograr tener iteraciones que no se cruzaran las unas con las otras se definió la duración de dichas iteraciones a una semana. Vale la pena aclarar que cada iteración idealmente debería contar con solamente un requerimiento (esto para mejorar los resultados), sin embargo, debido al espacio de tiempo limitado; necesidad de implementar 33 requerimientos en 11 semanas, esto fue imposible.

Para lograr cumplir objetivos, respecto a las fechas límites, fue necesario realizar iteraciones con más de un requerimiento, e independientemente de la aprobación o no del mismo, no se condicionó el comienzo de nuevas iteraciones con nuevos requerimientos. En casos donde fueran necesarias correcciones, se procedió a realizar una nueva iteración sobre los requerimientos que deberían presentar arreglos y en posteriores reuniones se realizó la reevaluación de las mejoras.

Inicialmente se crearon primitivas para representar estructuras definidas (primitiva clase, primitiva paquete, etc.) y el manejo de los principales gestos *multi-touch: drag, pinch, spread* [30]. Sin embargo, en este punto se concluyó con el cliente que dicha metodología de desarrollo pese a ser efectiva, impedía el crecimiento del *framework* al nivel esperado y lo limitaba al uso de estructuras gráficas definidas.

Después de aclarar dicha situación se procedió a reiniciar el proceso de desarrollo, se extrajo de la versión anterior la clase de manejo de gestos *multi-touch* y pese a que estos no fueron aprobados en su totalidad, se avanzó al respecto, a pesar de que eran funcionales no poseían el nivel de fluidez esperado.

Bajo la asesoría del cliente, se decidió darle una prioridad especial al desarrollo del módulo de lectura de especificación de diagramas de *software*. Para el desarrollo del módulo de lectura, fue usado el patrón de diseño *Facade* [44], el cual permite establecer una interfaz para diferentes formatos de especificaciones de diagramas, en este caso se implementó la interfaz para realizar lectura de formatos *.JSON*.

La clase que implementó la interfaz se encarga de dos tareas, en primer lugar realiza la lectura de la especificación del diagrama, luego a partir de esta información se encarga de guardarla en una instancia de una clase llamada zNode (para cada nodo incluido en la especificación se realiza una instancia de la clase), dicha instancia posteriormente es asignada a una instancia de clase zGNode la cual aparte de poseer la información de la especificación del diagrama para un nodo en específico, se encarga de controlar los comportamientos visuales del nodo; posición en pantalla, definir el contenedor, dibujar el objeto, etc.

Al contar con el canal de comunicación del flujo de información del *framework* armado y funcional (transferir información desde el archivo .JSON hasta la clase zNode) se procedió a agregar elementos a dicho canal, elementos a los que nos referiremos como primitivas. Inicialmente se agregó la primitiva rectángulo, esto permite que al visualizar en pantalla el diagrama se observe el rectángulo especificado en el archivo .JSON.

Para poder garantizar la escalabilidad de ZoomTI++ (más allá de la escalabilidad alcanzada con el uso del patrón *Facade* en los módulos de lectura), se creó una clase abstracta llamada *zPrimitive*, esta clase se encarga de agrupar atributos propios de toda primitiva; tipo de primitiva, rangos de *zoom* donde debe ser mostrada, color, entre otros. Adicionalmente, toda primitiva incluye un listado de otras primitivas (las cuales no son definidas sino hasta tiempo de ejecución), para poder abstraer dicha lista hasta la clase abstracta se hace uso de polimorfismo [45].

Usando el patrón *Abstract Factory* [44] se permite que toda primitiva extienda la clase abstracta y de igual manera facilita el proceso de creación de nuevas primitivas. Bajo esta lógica se permite la escalabilidad en un nuevo sector de *ZoomTI++*.

ZGNode necesita hacer uso de las primitivas guardadas de dos maneras: en primer lugar se necesita definir el contorno que rodea la figura (para el manejo de eventos, como colisión por ejemplo) y en segundo lugar para determinar qué tipo de figura se va a dibujar.

En este punto se alcanza una base sólida para el *framework* final, se puede cargar una especificación de un diagrama sencillo (nodos compuesto por rectángulos) e interactuar con ella en pantalla.

Antes de proceder a implementar nuevos requerimientos era necesario cerrar las iteraciones hasta este punto, porque los siguientes requerimientos eran en su mayoría sobre nuevas primitivas. En caso de ser necesario realizar algún cambio sobre alguna primitiva, se vería afectado desde su especificación hasta su muestra en pantalla.

Durante esta etapa de cierre de iteraciones tempranas del desarrollo fue necesario realizar diferentes ajustas sobre la especificación del diagrama, para permitir que una primitiva pueda contener nuevas primitivas (componente esencial del *zoom* semántico), esto implicó la implementación de algunos requerimientos de *layouts (layout* padre, *layout* hijo) para poder ubicar los nodos contenidos en posiciones relativas a su padre inmediato. Otro factor clave en la implementación de este aspecto fue el proceso de escalamiento de un nodo para poder ser representado visualmente al interior de su padre sin generar conflicto en la representación (disminución de su tamaño a un nivel aceptable).

Al recibir la aceptación por parte del cliente se continuó con la inclusión de nuevas primitivas, siempre respetando el patrón para esto. Este proceso requirió de bastante tiempo, sin embargo, al estar bien definido no presentó un desafió significativo, más allá de cuestiones propias de cada primitiva; determinar el centro de la elipse, etc.

En paralelo a este proceso se trabajaron otro tipo de requerimientos; requerimientos de navegación y requerimientos de aristas.

La navegación hasta este punto se realizaba a través de los gestos de *pinch*, *drag* y *spread*, para facilitar la velocidad de navegación del usuario se introdujo el reconocimiento de gestos *double tap*, con los cuales se puede centrar la vista en el elemento sobre el cual se aplica el gesto; centrar la vista involucra centrar la vista al nodo y modificar el nivel de *zoom* para que dicho nodo cubra la totalidad de la pantalla.

El mayor desafío para la implementación de este nuevo gesto radicó en la determinación del nivel de zoom necesario para que el nodo ocupe la totalidad de la pantalla; el valor está condicionado por el tamaño de la figura (por ende la forma de la figura también afecta este proceso) y el tamaño de la pantalla, adicionalmente este cálculo debe ser realizado durante el momento de construir el objeto para que la vista pueda ser modificada de acuerdo a la petición.

El otro elemento clave para representación de diagramas de software basados en grafos es el uso de aristas. Para conectar dos nodos es necesario determinar la posición de ambos nodos (la cual se encuentra en las propiedades de zGNode) y dibujar una línea de un nodo a otro. Sin embargo para poder realizar la unión de una forma visualmente correcta fue necesario para cualquier nodo determinar cuatro puntos de ancla, los que permiten realizar la unión con una línea lo más corta posible.

Las pruebas funcionales realizadas por el cliente en este punto arrojaron resultados positivos, sin embargo, el énfasis visual de la aplicación pese a estar correcto carecía de interactividad; funcionamiento muy plano. Para corregir este campo se pasó a realizar animaciones de las siguientes funcionalidades; animación del *double tap*, aparición y desaparición de aristas y nodos (al variar el nivel del *zoom*).

Para aumentar la variedad de opciones disponibles y la fluidez de la interacción con la aplicación se agregaron diferentes botones:

- Cargar modelo: Permite seleccionar otra especificación de diagrama, esto implica labores de liberación de memoria para crear nuevas instancias de las diferentes clases sin alcanzar un punto de volcado de pila.
- Ver modelo completo: Restaura la vista a su punto de inicio; centra la vista en las coordenadas iniciales y restaura el nivel de *zoom* al valor inicial. En este punto el dominio sobre las transformaciones básicas del *canvas* es lo suficientemente alto para hacer funcionar este botón sin mayores complicaciones.
- Regresar un nivel: Modifica la vista; centrado y nivel de *zoom*, al nodo inmediatamente superior (en caso de no haber tal nodo, se restaura el nivel de *zoom* al inicial centrando al vista en el nodo con visión actual). Este botón presentó complicación en el sentido

de que se vuelve necesario determinar dinámicamente que nodo se encuentra "enfocado". Para realizar dicha labor se determinó el nodo más cercano respecto al centro de la pantalla y se determinó el padre de dicho nodo, lo demás es proceder con transformaciones tal como ya se ha venido indicado.

Conservando el principio de que la apariencia, la fluidez y la realimentación continua del usuario son esenciales para el éxito de *ZoomTI++* se incorporó animaciones a cada uno de estos botones, de esta manera el usuario permanece actualizado de qué tipo de acción se ejecuta con sus decisiones.

Bajo recomendación del cliente, para detección de nuevos errores, se procedió a creación de ejemplos que exigieran a un nivel más alto la aplicación. Esta labor llevó varios ciclos completos en los que se detectaban errores pequeños (información no siendo mostrada de forma clara, movimiento no tan fluido, botones no funcionando de la forma esperada, etc.) y se iban corrigiendo.

En este punto la implementación del *framework* estaba aproximadamente en un 85%. Originalmente se contaba con un número diferente de requerimientos, pero debido a las necesidades (ya mencionadas) que iban apareciendo durante el transcurso del desarrollo, se fue refinando la especificación de los requerimientos para incluirlos; ya sea para desarrollo durante el ciclo, botones o para trabajo futuro, *layout* de aristas.

La etapa siguiente consistió en implementación de los *layouts* que no fueron declarados fuera del alcance del proyecto. Debido a que independientemente del tipo de layout a implementar se puede traducir todo como realizar una translación de los nodos de la ubicación actual a una ubicación destino, se aplicó el patrón *Strategy* [44] el cual plantea un número de algoritmos para la reubicación de los nodos, partiendo del principio ya expuesto, de igual manera se permite realizar variaciones donde el algoritmo cuenta o no con la animación (dependiendo de la necesidad).

La implementación de la lógica detrás de los *layouts* no fue compleja, sin embargo es importante destacar el comportamiento que debe tener un nodo al contener un listado de más nodos; estos deben actualizarse de la manera correcta para desplazarse con su padre (en caso de que este se vea afectado por la ejecución de dichos *layouts*). Para lograr este efecto se hizo uso de una variación del patrón *Observer* [44] mediante el cual el padre se encarga de notificar a sus hijos del cambio en ejecución para que estos a su vez expongan dicho cambio a sus hijos, en forma recursiva y de esta manera mantener la integridad de la vista.

La etapa final de la construcción se concentró en tres requerimientos; primitivas vectoriales (carga de imágenes para usar como nodos), reorganización *zoom* semántico (ejecutar *layouts* por defecto, según la especificación o en cierto nivel de *zoom*) y el manual de usuario/programador.

El manual de usuario/programador se organizó en las dos secciones referentes a su nombre; la sección de usuario presenta la manera como la cual un usuario final puede hacer uso de las diferentes funcionalidades de *ZoomTI++*, esto incluye pero no se limita a:

- Descripción de cada uno de los gestos táctiles aceptados. Nombre, descripción y manera de usarlo.
- Descripción de cada uno de los botones disponibles. Nombre, descripción y momentos adecuados para usarlos.

El énfasis para programador se divide en dos secciones: inicialmente se presenta una descripción formal de la estructura genérica de la especificación del diagrama en el archivo JSON. A continuación se mencionan los elementos explicados en esta sección:

El formato del nodo contiene los siguientes elementos:

- Id: Identificador único que es un valor real del nodo.
- x, y: correspondientes a las coordenadas en el canvas del nodo y son valores reales.
- color: que es el color de contorno que puede estar representado en colores en inglés o con el código hexadecimal
- fillColor: Referente al color de relleno del mismo que puede estar representado en colores en inglés o con el código hexadecimal, si se quiere transparencia, se pone como valor "transparent"
- sameLineText: Define si el texto del nodo está en una o varias líneas. Si es "true " reorganiza el texto en varias líneas y de ser "false" lo hace en una sola.

El nodo a su vez contiene n primitivas que a su vez pueden contener m primitivas dentro de ellas denotado con contains.

Todas las primitivas tienen los atributos

- type: referente a su tipo que es un valor entero: 1 para rectángulo, 2 para elipses, 3 para textos, 4 para líneas paralelas y 5 para imágenes
- x, y : Las posiciones relativas al nodo padre que puede ser un valor real.
- zoomStart y zoomEnd: Los valores reales relativos en los cuales el nodo aparece y desaparece respectivamente de la escena.
- rows (opcional): Representa la cantidad de filas del layout matricial, siendo un valor entero.
- columns (opcional): Representa la cantidad de columnas del layout matricial, siendo un valor entero.
- defaultLayout (opcional): El layout por defecto a mostrar del nodo, "circle" o "matrix".
- customLayout (opcional): El layout determinado a un nivel de zoom especificado en el JSON, "circle" o "matrix".
- customZoom (opcional): El nivel de zoom necesario para que se active el custom layout. Siendo un valor real.

Adicionalmente a estos valores genéricos, cada una de las primitivas incluye una especificación de elementos propios a cada una, por ejemplo, la primitiva rectángulo espera valores de ancho y alto. Estos elementos varían entre primitivas.

La segunda sección del módulo de programador presenta la manera de agregar nuevas funcionalidades a *ZoomTI++*; indicando las clases y métodos a implementar, mostrando un ejemplo de cada una de estas labores.

Actualmente el manual soporta las instrucciones para agregar: nuevos *layouts*, nuevas primitivas y nuevos gestos.

Para el momento en el que estos requerimientos fueron implementados y con esto, alcanzado el 100% de avance en la implementación, se poseía la aprobación del cliente en la totalidad de iteraciones realizadas hasta ese punto.

La finalización de esta fase radicó en la creación de ejemplos completos, los cuales permitieran evaluar la mayor parte de las funcionalidades de *ZoomTI++* (fueron usados durante el video de demostración).

VI – RESULTADOS

De forma adicional a este documento memoria, en la propuesta se indicaron varios entregables como los esperados durante el desarrollo del trabajo de grado:

Prototipo de plataforma compuesto de:

- Especificación de requerimientos.
- Descripción de la arquitectura del sistema.
- Código fuente.
- Manual de uso de la plataforma
- Documento de pruebas de aceptación.

Una de las principales razones para realizar iteraciones cortas era para recibir evaluación por parte del cliente, principal interesado en el *framework*, como se indicó en la metodología, la aprobación de *ZoomTI++* está condicionada por los resultados de las pruebas de aceptación.

Es importante resaltar el hecho de que uno de los requerimientos no funcionales hacía referencia al rendimiento del *framework*, exigiendo que este superara las pruebas determinadas durante el proceso de selección de herramienta base que determinó a *QT* como la indicada, este fue uno de los criterios usados por el cliente para aceptar o rechazar los requerimientos durante las iteraciones.

El resultado más grande que reúne todos los objetivos del trabajo de grado es la muestra del *framework* y sus funcionalidades, para esto se presentan ilustraciones con su respectiva descripción.

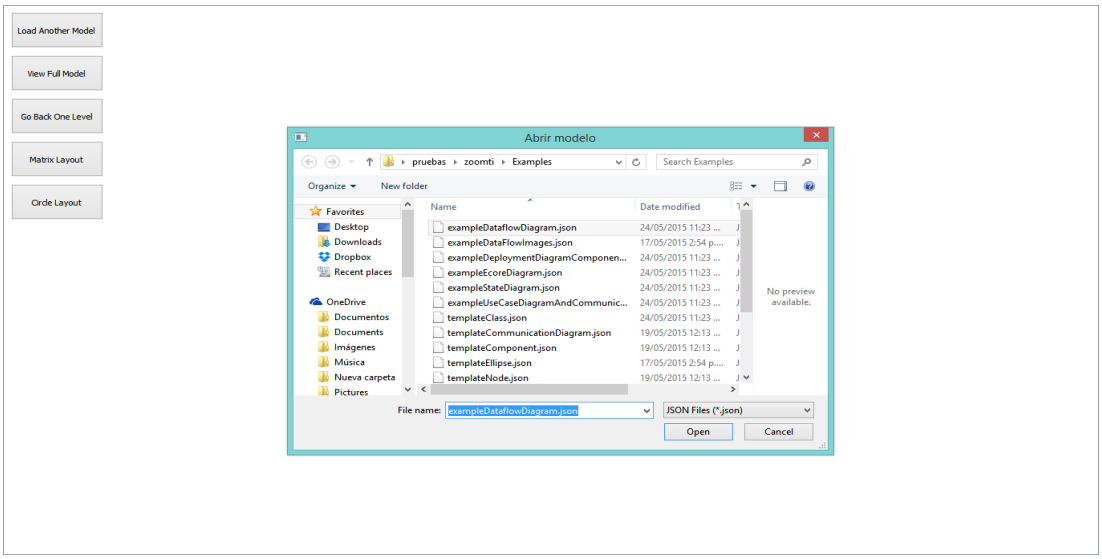


Ilustración 3 Carga de modelo

Al ejecutar el *framework* por primera vez o al acceder por medio del botón "*Load Another Model*" se presenta una pantalla de selección de archivos, esperando seleccionar un archivo con extensión .JSON y la especificación correcta del diagrama.

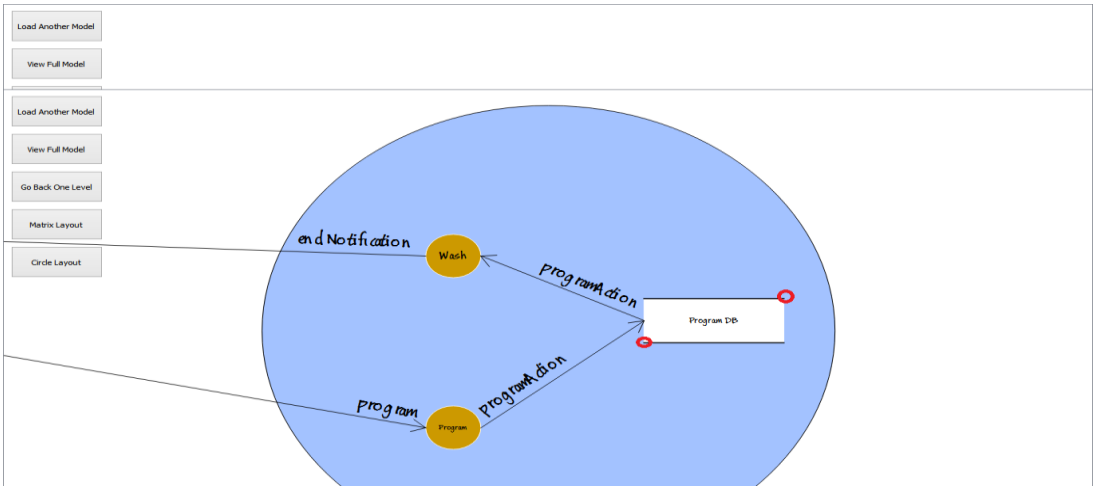


Ilustración 4 Estado previo al Zoom-Out

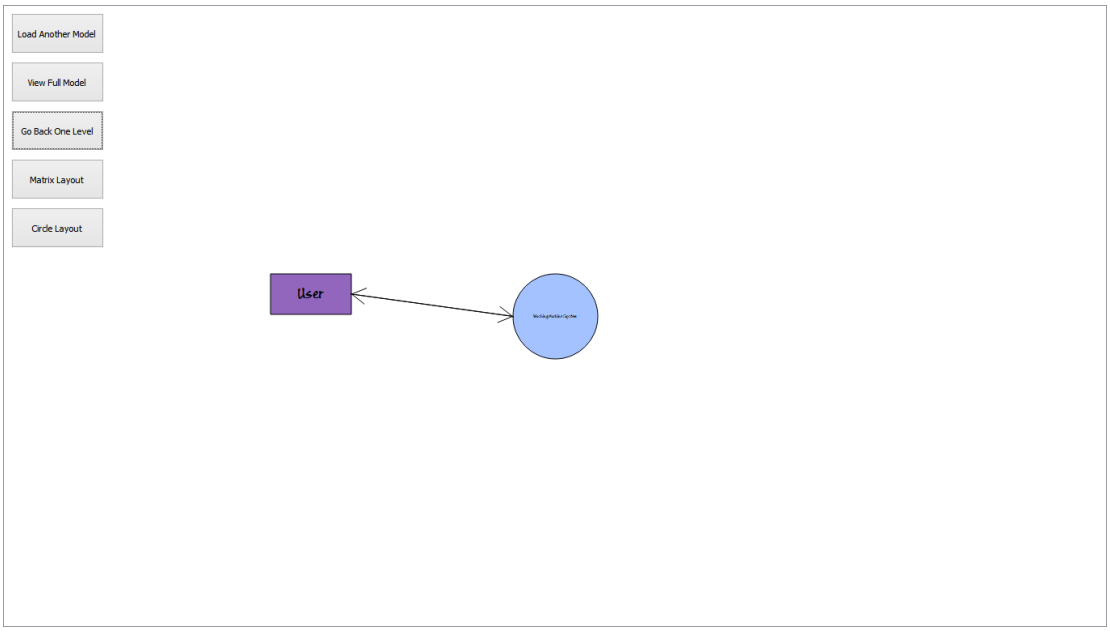


Ilustración 5 Estado posterior al Zoom-Out

Es difícil mostrar el reconocimiento de gestos *multi-touch* a través de imágenes, el video de demostración soluciona dicha situación. Para aumentar o disminuir el nivel de *zoom* sobre cualquier punto se realiza *spread* o *pinch* respectivamente, las imágenes muestran un ejemplo de *pinch [30]*.

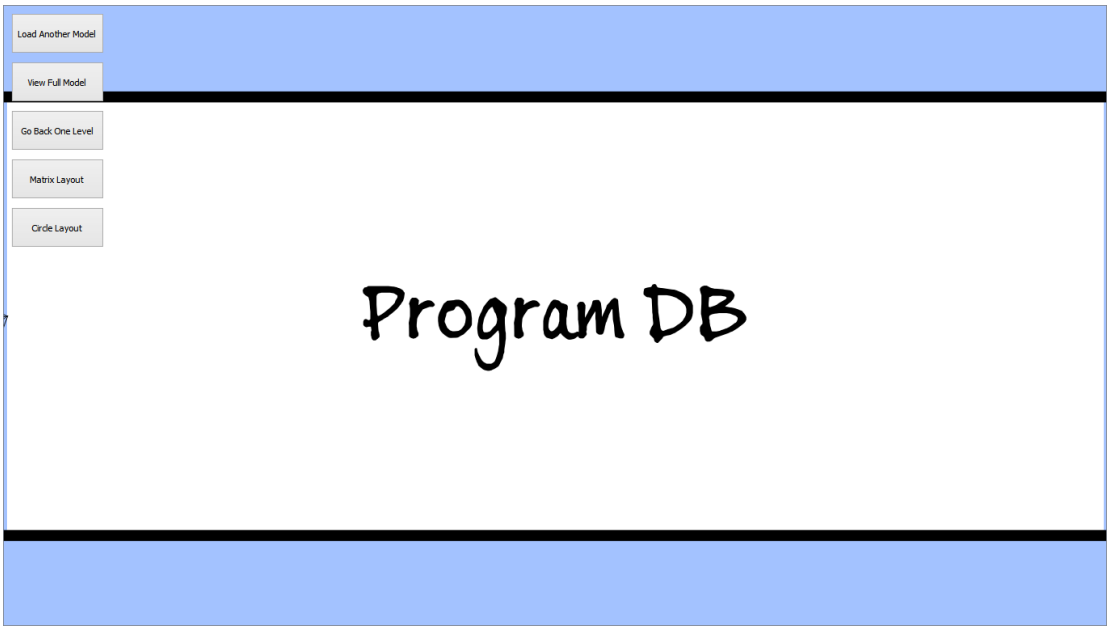


Ilustración 6 Estado posterior al Zoom-In

La imagen anterior muestra el comportamiento después de realizar *spread*.

Vale la pena resaltar que estos gestos son aceptados de forma general y son usados en un alto número de dispositivos.

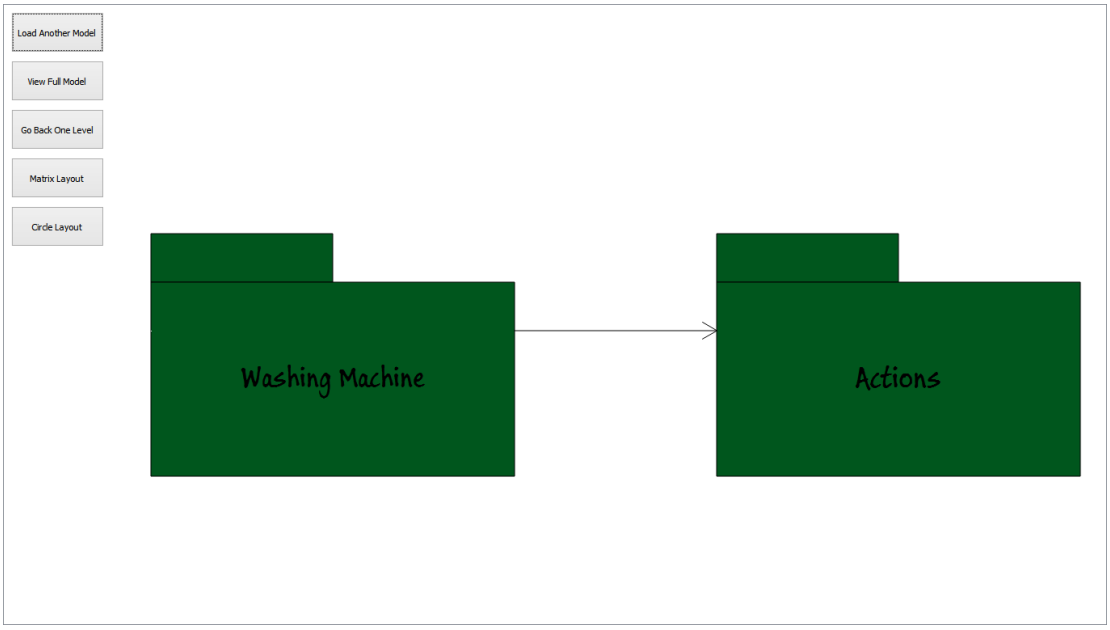


Ilustración 7 Estado previo al Doble clic – Double Tap

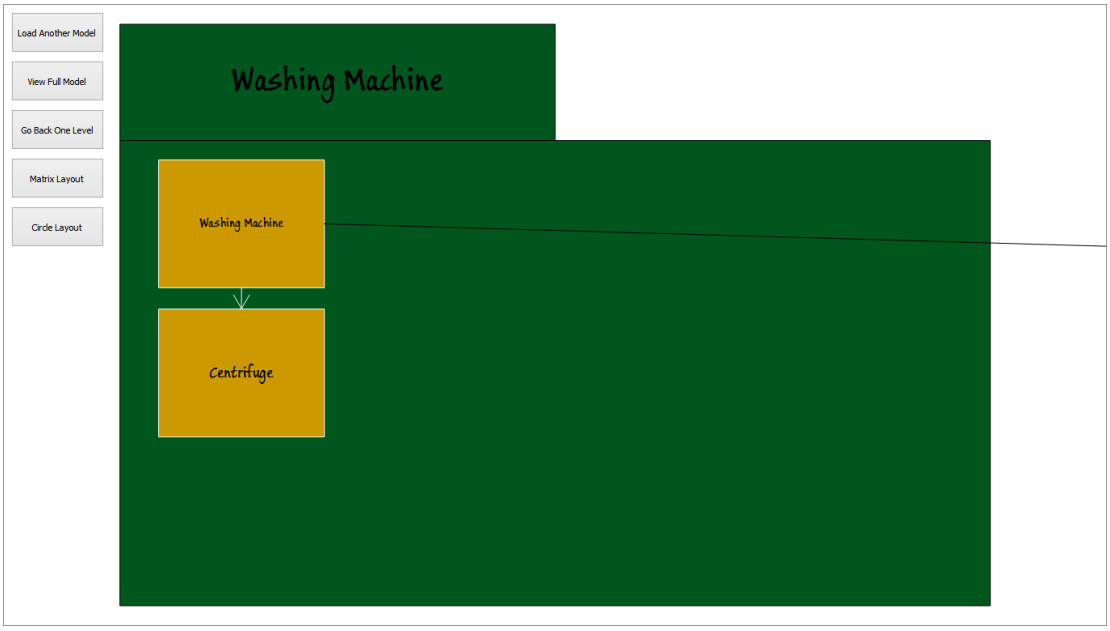


Ilustración 8 Estado posterior al Doble clic – Double Tap

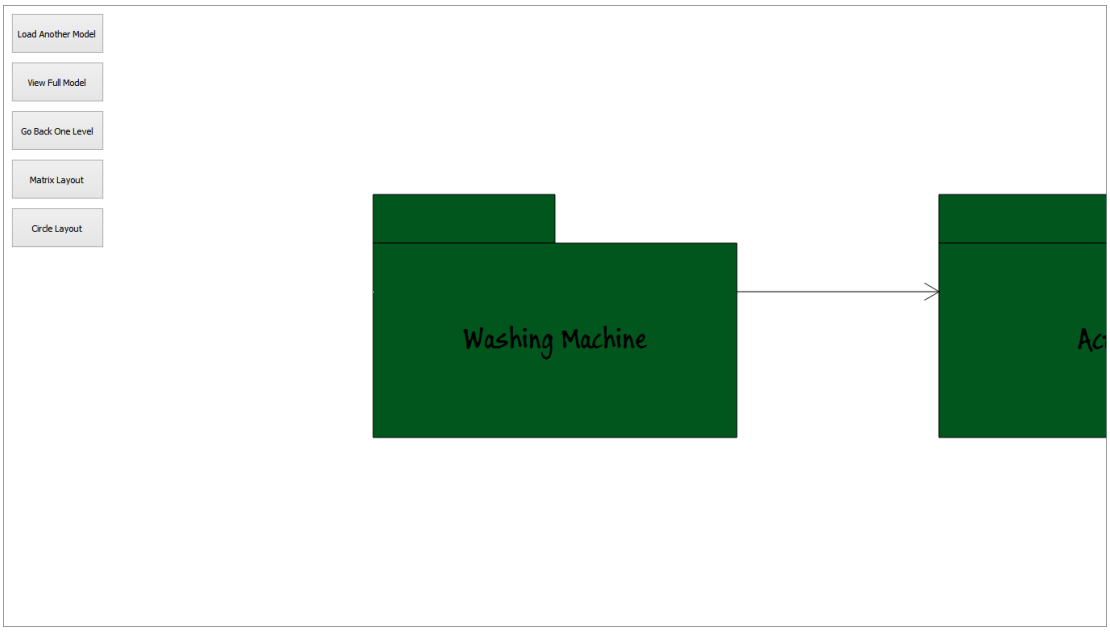


Ilustración 9 Estado posterior al Doble clic – Double Tap sobre un elemento con zoom máximo

Otro de los gestos reconocidos es el *double tap* [30] el cual representa una simulación de un doble clic sobre la pantalla, al realizar dichos gestos sobre la pantalla se puede realizar "enfoque" sobre un nodo llevando la vista a mostrar dicho elemento en su nivel máximo (ocupando toda la pantalla). Al momento de repetir el gesto sobre un elemento cuya representación es actualmente la más alta, se regresa un nivel de *zoom*, como lo muestra la última imagen.

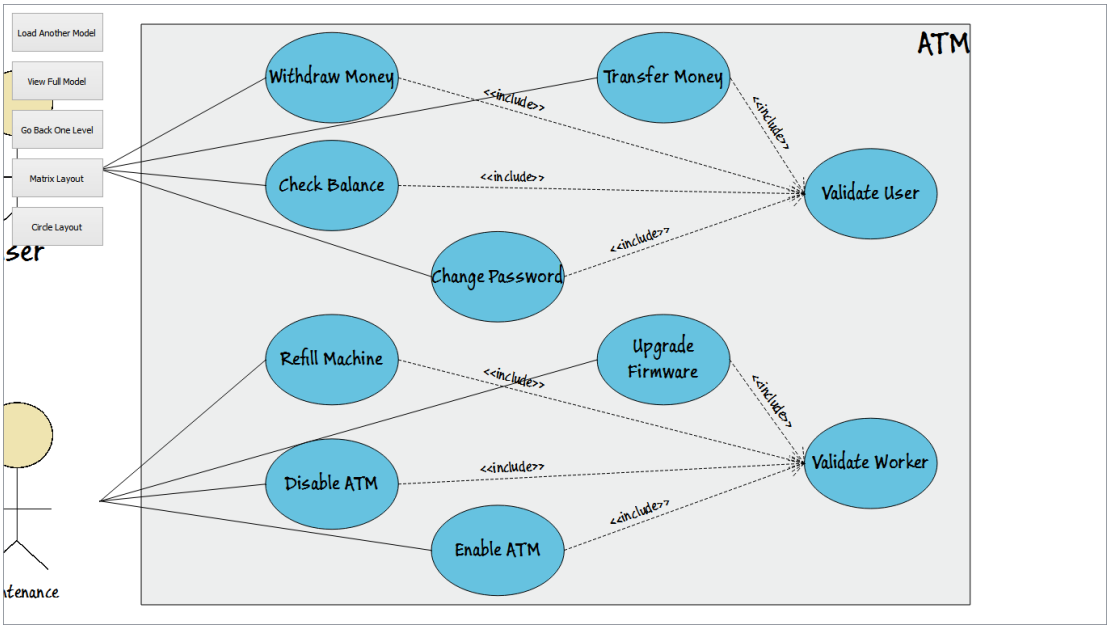


Ilustración 10 Estado previo a la aplicación de layout

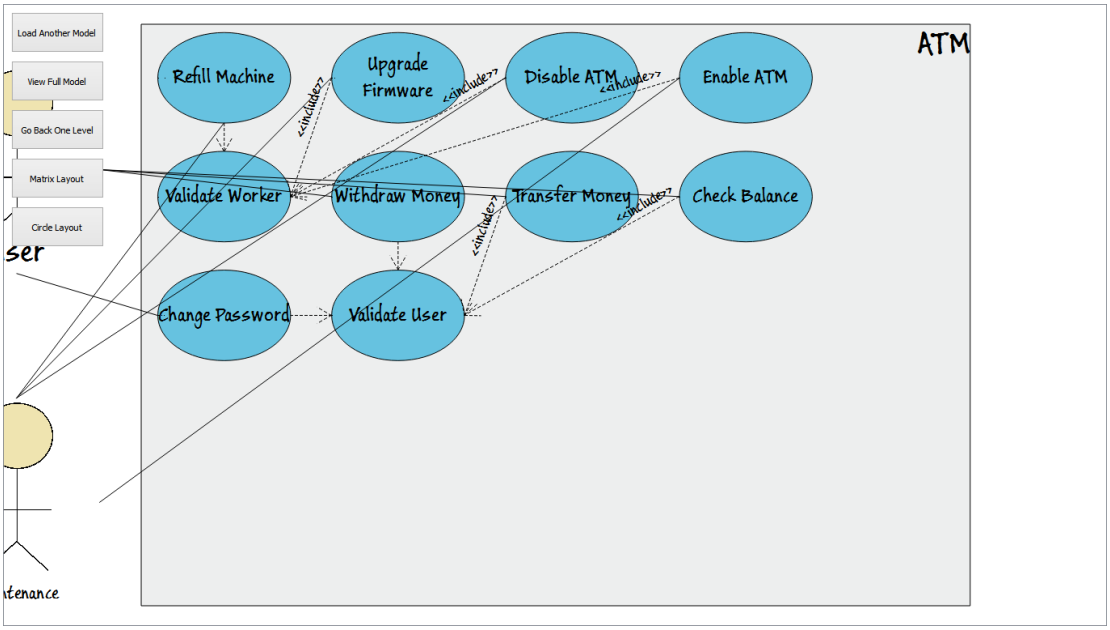


Ilustración 11 Resultado ejecución layout matricial

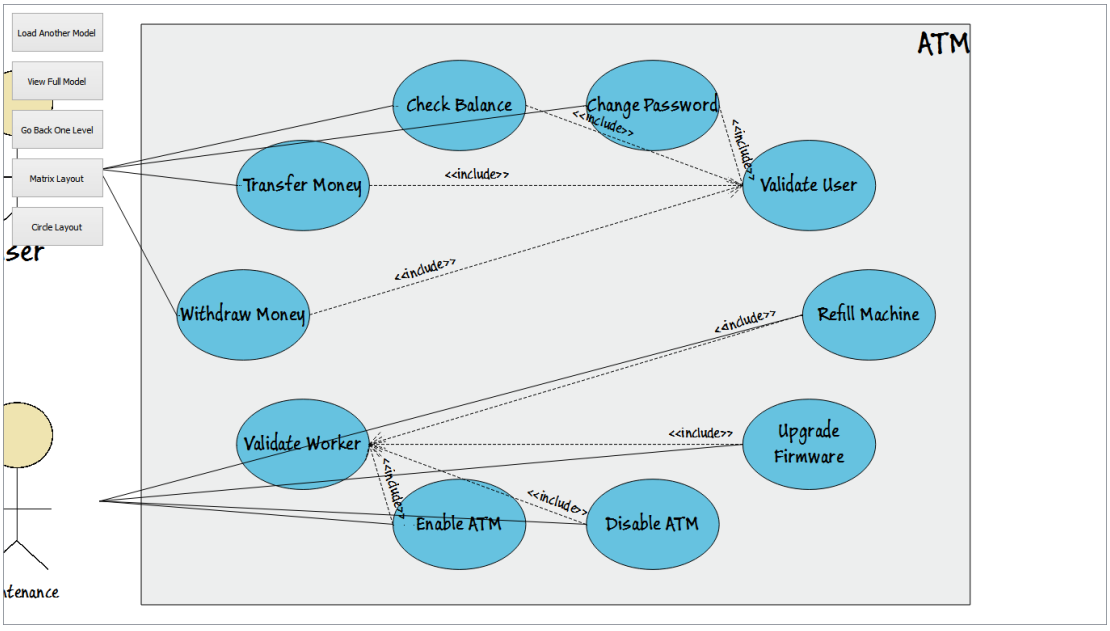


Ilustración 12 Resultado ejecución layout circular

La aplicación de *layouts* se realiza por medio de los botones a disposición del usuario en todo momento y son aplicados al conjunto de elementos contenidos en el nodo sobre el cual hay énfasis en el momento de solicitar la ejecución.

Dicha ejecución posee una animación para mostrar el proceso de reorganización al usuario.

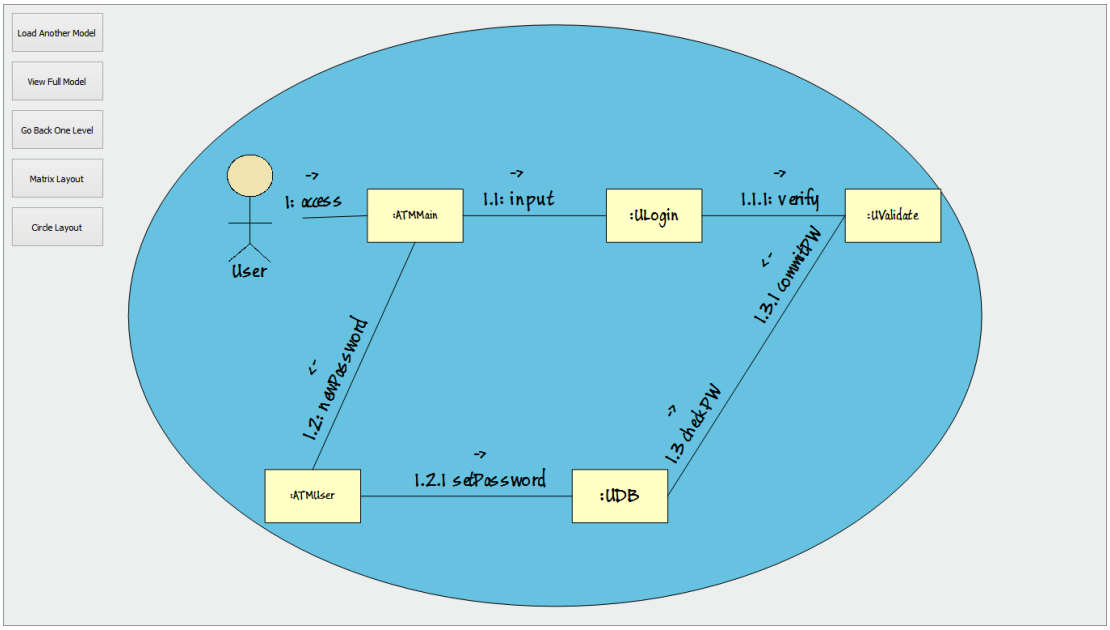


Ilustración 13 Estado previo a "Go Back One Level"

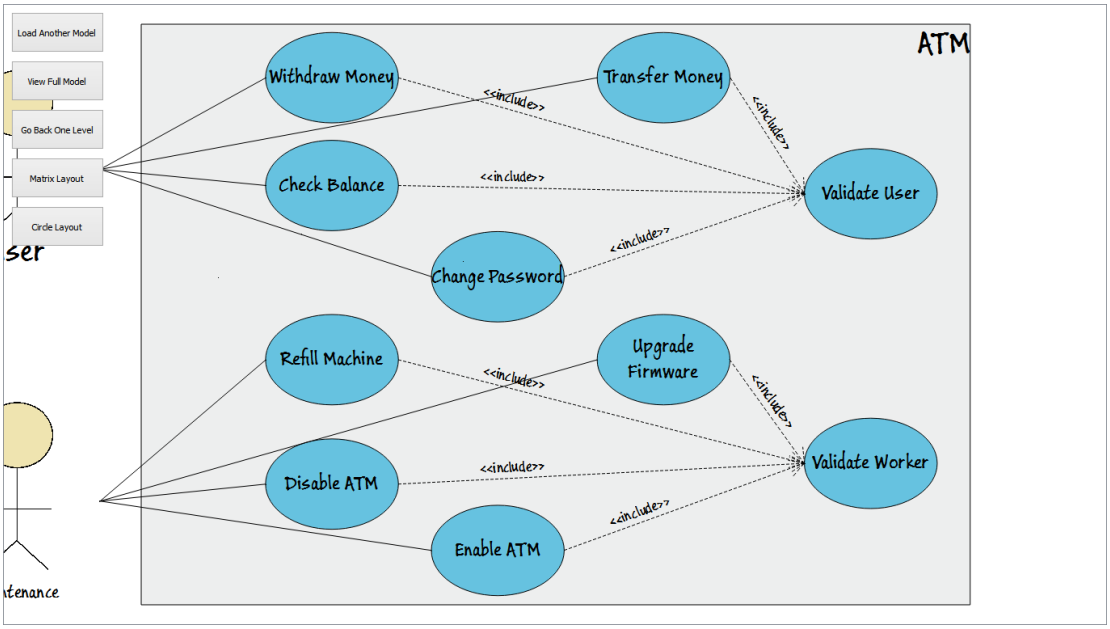


Ilustración 14 Estado posterior a "Go Back One Level"

Las figuras presentan el uso de otro de los botones el cual permite realizar saltos de navegación en el diagrama entre nodos y sus padres, a través de una animación; se pasa del nodo con el

"enfoque" actual al nodo padre de este, en caso de que lo tenga, caso contrario restaura la vista a la vista por defecto.

Ilustración 15 Estado previo a "View Full Model"

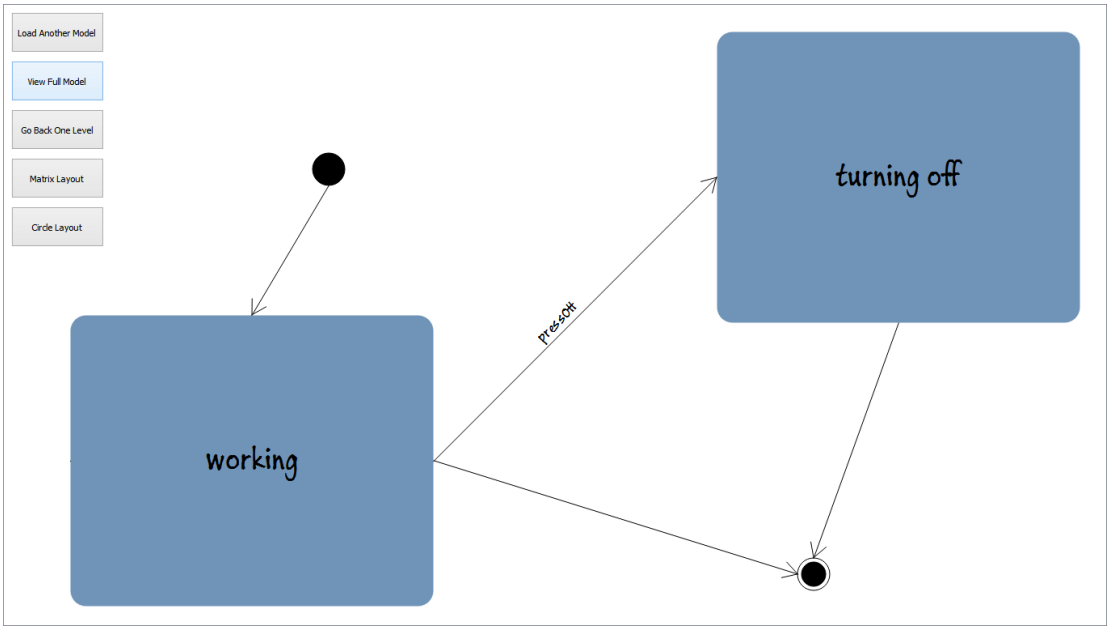


Ilustración 16 Estado posterior a "View Full Model"

Este botón permite una manera directa para pasar de cualquier nivel de detalle y cualquier posición sobre el *canvas*, al centrado del *canvas* con nivel de *zoom* inicial. Dicha transición se realiza con una animación.

Finalmente con respecto a las pruebas de aceptación, se diseñó una plantilla de prueba para evaluar los diferentes requerimientos del sistema, a continuación se presenta la plantilla seguida durante este proceso.

FORMATO DE PRUEBA DE ACEPTACIÓN	
Num Requerimiento(s):	*Tomado de la especificación de requerimientos [1]*
Descripción Requerimiento(s):	*Tomado de la especificación de requerimientos [1]*
Objetivo de la prueba:	*Descripción breve de que se busca probar*
Descripción de la prueba:	*Descripción breve del procedimiento que se va a seguir, sin entrar en detalle*
PRECONDICIONES	
1	*Condiciones que deben estar en juego antes de ejecutar la prueba*
2	
3	
PASOS DE LA PRUEBA	
1	*Descripción detallada de los procedimientos a seguir para la aplicación de la prueba, se recomienda apoyarse de imagenes*
2	
3	
RESULTADOS ESPERADOS	
DESCRIPCIÓN	***IMAGEN***
Descripción del resultado mostrado en la imagen para destacar los elementos que se deben verificar	
RESULTADOS OBTENIDOS	
DESCRIPCIÓN	***IMAGEN***
Observaciones encontradas a realizar la prueba	

Resultado de la Prueba:	*Aprobada o Fallada*

Tabla 7 Plantlla prueba de aceptación

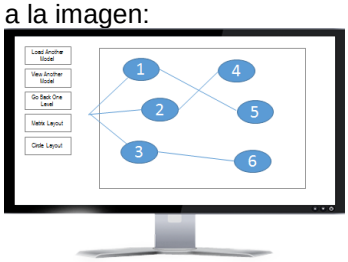
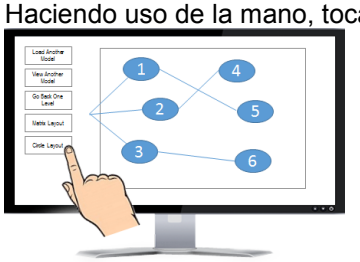
A partir de la plantilla se probaron los diferentes requerimientos, la siguiente tabla presenta los resultados en forma de totales. Estos totales son considerando 27 requerimientos (quitando los 6 requerimientos no funcionales) y a partir de una clasificación informal de los requerimientos. No se toma en cuenta el número de pruebas, sino el número de requerimientos que han sido evaluados con estas.

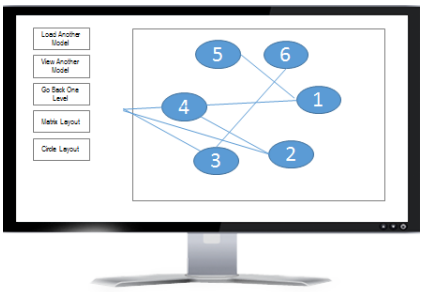
Tipo de requerimientos	#Requerimientos		
	Total	Probados	Aprobados
Lectura de la especificación del diagrama	4	4	4
Multi-Touch	5	5	5
Layouts	4	4	4
Nodos y Aristas	9	9	9
Zoom Semántico	5	5	5

Tabla 8 Totales Pruebas

Finalmente se presenta un ejemplo de las pruebas, el formato de pruebas referente al requerimiento 18 el cual hace referencia a la aplicación de *layout* circular.

FORMATO DE PRUEBA DE ACEPTACIÓN	
Num Requerimiento(s):	18
Descripción Requerimiento(s):	18 - El *framework* debe organizar los elementos de grafos con layouts circulares.
Objetivo de la prueba:	Validar la aplicación del *layout* circular
Descripción de la prueba:	Se realiza una guia para mostrarle al usuario la manera en la cual debe aplicar el *layout* en cuestión.
PRECONDICIONES	
1	Contar con la carpeta de ejemplos en el equipo que ejecutará la prueba (viene incluida en el repositorio de *ZoomTI++*).
2	El *framework* no se debe encontrar en ejecución.
PASOS DE LA PRUEBA	

1	Ejecutar el *framework*
2	Al ejecutar, debe aparecer la pantalla de selección de archivo
3	Buscar la carpeta “Examples” y seleccionar el archivo “exampleUseCaseDiagramAndCommunication.json”
4	Tocar en Abrir/Open
5	En el diagrama que se muestra en pantalla, realizar *Double Tap* (Doble contacto de forma rápida) en el interior del rectángulo “ATM”, sin tocar uno de los elementos azules.
6	El diagrama mostrado en pantalla, después de realizar el paso 5, debe ser similar a la imagen: 
7	Haciendo uso de la mano, tocar el botón “Circle Layout”, ver imagen: 

RESULTADOS ESPERADOS	
DESCRIPCIÓN	***IMAGEN***
La imagen muestra la disposición de los elementos después de aplicar el *layout*. Se puede observar que el elemento con menor identificador en el grupo, es ubicado en el centro y en la derecha, desde ese punto a distancias equidistantes son ubicados los demás elementos, siguiendo las manecillas del reloj.	

RESULTADOS OBTENIDOS	
DESCRIPCIÓN	***IMAGEN***
Se muestra la pantalla obtenida al realizar los pasos previos al número 5.	

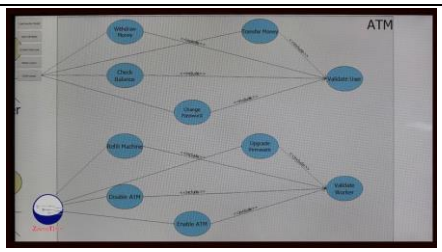
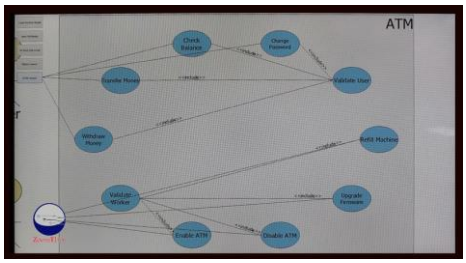
	
Al realizar el paso 6 se obtienen los resultados mostrados en la imagen. Los elementos son organizados en forma circular y al igual que en la imagen de los resultados esperados, conservan las relaciones entre ellos (representadas por aristas).	
Resultado de la Prueba:	**Aprobada**

Tabla 9 Aplicación Prueba Test Layout Circular

Con esto se da por finalizado el proceso de desarrollo.

VII – CONCLUSIONES

1. Análisis de Impacto del Desarrollo

El *framework* finalizado y funcional tendrá influencia en varios campos, tal como se presenta a continuación:

Desde un punto de vista económico se parte del supuesto que el tiempo es dinero, una de las ventajas de la visualización de diagramas de *software* a través de *zoom* semántico es eliminar la necesidad de diagramas en archivos individuales (como ya se presentó, una de las posibles causas de errores). A partir de este hecho, disminución de errores, y de mejoras en la calidad de los diseños, el tiempo en esta etapa crítica podría disminuir generando un ahorro de dinero. Hecho reforzado por la teoría de proyectos de *software* la cual muestra el costo de un error en una etapa avanzada del proyecto [46].

Desde un punto de vista social, el impacto de la solución se encuentra limitado debido a que los usuarios potenciales de *ZoomTI++* son personas trabajando en proyectos que incluyan el uso de modelos de *software* y que busquen visualizar la información de dichos modelos a través de diagramas. Dicha etapa del proyecto suele pasar desapercibida, desde un punto de vista de un usuario final de un producto de *software*. Sin embargo desde el contexto social de desarrolladores y trabajadores en proyectos de esta naturaleza, el impacto se centra en permitir hacer uso del *zoom* semántico y de la interacción con la pantalla *multi-touch* para mejorar la comprensión de la relación entre diagramas de diferente tipo, lo cual podría permitir una mejora en la calidad del diseño de modelos.

Finalmente desde un punto de vista disciplinar, el impacto más evidente para los usuarios de ZoomTI++, debido a que fue concebido como un componente parte de una potencial herramienta CASE, es la posibilidad de crecimiento del *framework* , el cual puede ser ajustado a las necesidades del usuario.

El diseño está contemplado para brindar la posibilidad de crecer en tres aspectos. En primer lugar existe la posibilidad de soportar diagramas en formatos diferentes al usado actualmente *(.JSON*), esto elimina la restricción de tener que brindar diagramas solamente en este formato. Se brinda una interfaz la cual debe ser implementada por el nuevo sistema de diagramas de entrada. En segundo lugar, la posibilidad de agregar nuevas primitivas para ser representadas gráficamente, se presenta una clase abstracta con los elementos bases de toda primitiva, al extender dicha clase se hace adquisición de propiedades básicas; rangos de *zoom*, posiciones, entre otros. Finalmente también existe la posibilidad de agregar nuevos *layouts*, se presenta una clase abstracta con el comportamiento para realizar el traslado de nodos, haciendo uso o no de una animación.

2. Conclusiones y Trabajo Futuro

La elección de *QT* como base para el desarrollo de *ZoomTI++* fue adecuada, debido a la orientación de *QT* para el desarrollo de aplicaciones con interfaz gráfica, a su buen manejo de liberación de memoria para elementos gráficos y a sus facilidades para el manejo de eventos y señales. Cada una de estas características fueron usadas de principio a fin en *ZoomTI++* y gracias a *QT* su uso fue sencillo y adecuado.

El alcance definido en las primeras etapas fue acertado, se identificaron 39 requerimientos (descartaron 6 en la priorización) y se procedió a implementar los 33 restantes. Cada uno de estos requerimientos fueron implementados y las estrategias usadas para disminuir el riesgo de sobredimensionar el alcance del trabajo de grado fueron correctas para alcanzar ese estado de finalización. Esto se debió de igual forma al uso de correctas prácticas de programación las cuales ayudaron a la integración de nuevas funcionalidades con el avance del tiempo de desarrollo.

La importancia de definir una arquitectura adecuada para *ZoomTI++* fue un aspecto clave en el éxito del desarrollo, dicha arquitectura permitió cumplir con los estándares de calidad esperados y funcionó de guía durante el proceso de desarrollo. Gracias a esta y a escoger una metodología de desarrollo adecuada; *RUP* y sus ventajas; contar con pasos adaptables a la facilidad del equipo de desarrolladores, reflejado en iteraciones cortas de desarrollo con revisiones continuas del cliente para cada requerimiento implementado, con la posibilidad de reiterar sobre un requerimiento de ser necesario.

La ventaja principal de iteraciones cortas (semanales, terminando cada iteración con una reunión), fue recibir la realimentación por parte del cliente para la realización de correcciones prácticamente inmediatas, esto evitó que en etapas posteriores se repitieran errores y a su vez esto desencadenara un crecimiento en el tiempo de corrección.

Finalmente, escoger C++ como lenguaje para el desarrollo de *ZoomTI++* y gracias a que *QT* es escrita de igual manera en este lenguaje, fue un elemento clave para alcanzar niveles de

rendimiento lo suficientemente adecuados para superar las pruebas definidas al comienzo del proyecto. De igual manera en forma comparativa con aproximaciones similares desarrolladas en el pasado, se puede observar la fluidez en el comportamiento del *framework*, en situaciones donde otros presentarían perdida de rendimiento. Para alcanzar este punto fue necesario hacer uso de algoritmos y estructuras de datos adecuadas (para no convertir labores realizadas por el equipo de trabajo en factores que generarán perdida de rendimiento).

Debido a la naturaleza del *framework*, ser componente de una herramienta *CASE*, el futuro de *ZoomTI++* se puede concentrar en tres grandes campos; Mejora de lo implementado, implementación de requerimientos descartados (incluyendo nuevos requerimientos) e integración con componentes diferentes de la herramienta *CASE* completa.

A pesar de que *ZoomTI++* se encuentra completo; se implementaron la totalidad de los requerimientos detectados después de realizar la priorización (33), hay oportunidades para mejorar la implementación realizada. Algunas de las funcionalidades pese a que poseen un comportamiento adecuado, no se encuentran en un estado ideal; por ejemplo el proceso para determinar si un nodo debe ser dibujado o no, hace uso del nivel de detalle de la pantalla (LOD), pero la manera en la que se encuentra codificado, cada elemento tiene un LOD independiente (dependiendo de su tamaño en relación a la pantalla después de ser escalado), para mayor control del usuario, dicho valor debería ser traducido a valores de la transformación completa de la pantalla.

El identificar un elemento donde la falta de normalización es una causa de problemas lleva a proponer que una posibilidad de mejora significativa para *ZoomTI++* es normalizar los valores usados en las primitivas, comenzando por definir un LOD genérico para todo el modelo y de esta manera estandarizar a un punto más alto lo presente y a su vez permitir que el ingreso de nuevas primitivas sea estándar, en relación a los tamaños y proporciones.

Otra oportunidad de mejora de lo ya implementado es el comportamiento de las primitivas texto, dichas primitivas actualmente es necesario que se encuentren al interior de otra primitiva y esto a su vez imposibilita su animación (de aparición y desaparición). Es recomendable simplificarlas al extraerlas a otra clase del tipo *ZG(ZoomTI++ Graphic)*.

Los requerimientos descartados se concentran en requerimientos de *layouts [47]*. Estos hacen referencia a no solo hacer uso del *zoom* semántico para variar la información en pantalla, sino a partir de este mismo principio, reorganizar los elementos en pantalla. *ZoomTI++* incluye cuatro tipos de *layouts* en su implementación: padre, coordenadas, circular, matricial.

Los *layouts* que se descartaron durante la priorización son: Jerarquico, ortogonal, árbol, posicionamiento, nodos adyacentes y aristas. Este último tiene una importancia particular, es el único *layout* específicamente orientado a aristas, los otros requerimientos se concentran en nodos.

Estos fueron los identificados en la especificación de requerimientos [48], sin embargo existen diferentes tipos de *layouts* que podrían ser implementados. De igual manera existen diferente tipo de primitivas las cuales pueden ser implementadas, por ejemplo más tipos de polígonos, uniones entre primitivas (mezclar una elipse y un triángulo), etc. Cada uno de estos aspectos

supone un desafío interesante para alguien que busque aumentar la complejidad de acciones que *ZoomTI++* puede realizar.

Saliendo del enfoque de *ZoomTI++*, un nuevo componente de la herramienta *CASE* puede ser implementado y hacer uso de las funcionalidades del *framework*, recordando que este trabajo es totalmente hacia la representación visual de diagramas de *software*, los demás componentes de una herramienta completa faltarían por ser codificados o conectados. Uno de estos componentes es el componente de automatización de creación de especificación de diagramas, actualmente para poder hacer uso de *ZoomTI++* se espera recibir una especificación de diagrama en formato *.JSON* siguiendo las guías que se encuentran en el manual de plataforma [49]. Dicha especificación actualmente debe ser generada manualmente y está en riesgo de cometer un alto número de errores. El componente de automatización se encargaría de evitar que el usuario cometa esas fallas.

Una manera óptima para permitir que *ZoomTI++* sea usado por diferentes usuarios es hacer uso de la intervención de un gestor de proyectos para que sea construido, probado y empaquetado, por ejemplo *CMAKE [*50*]*, lo cual permita en un futuro contar con las funcionalidades de *ZoomTI++* con solo incluir algunas líneas en su gestor de proyectos.

VIII - REFERENCIAS Y BIBLIOGRAFÍA

1. J. A. Pavlich-Mariscal, H. D. Veliz-Quispe, S. Demurjian, and L. D. Michel, "*Un ambiente de Meta-Modelado y Visualización Basado en el paradigma de Zoomable User Interfaces*," *Ingeniare*, 2014.

2. K. Hornbæk, B. B. Bederson, and C. Plaisant, "Navigation Patterns and Usability of Zoomable User Interfaces with and Without an Overview," *ACM Trans. Comput.-Hum. Interact.*, vol. 9, no. 4, pp. 362–389, Dec. 2002.

3. Cockburn, A. Karlson, and B. B. Bederson, "A review of overview+ detail, zooming, and focus+ context interfaces," *ACM Computing Surveys (CSUR)*, 2008.

4. D. Marinos, C. Geiger, T. Schwirten, and S. Göbel, "Multitouch Navigation in Zoomable User Interfaces for Large Diagrams," in *ACM International Conference on Interactive Tabletops and Surfaces*, New York, NY, USA, 2010, pp. 275–276.

5. T. T. A. Combs and B. B. Bederson, "Does zooming improve image browsing?," *Proceedings of the fourth ACM conference …*, 1999.

6. C. Chen, "Individual differences in a spatial-semantic virtual environment," *Journal of the American Society for Information …*, 2000.

7. D. O. Case, *Looking for information: A survey of research on information seeking, needs and behavior*. books.google.com, 2012.

8. F. Budinsky and S. A. Brodsky, *Merks, Eclipse Modeling Framework*. Pearson Education, 2003.

9. D. Steinberg, F. Budinsky, and M. Paternostro, *Merks, EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2009.

10. K. Balasubramanian, A. Gokhale, and G. Karsai, "Developing applications using model-driven design environments," *Computer*, 2006.

11. B. Moore, *Eclipse development using the graphical editing framework and the eclipse modeling framework*. IBM, International Technical Support …, 2004.

12. J. Kienzle, W. A. Abed, and J. Klein, "Aspect-oriented multi-view modeling," *Proceedings of the 8th ACM international …*, 2009.

13. P. Baker, Z. R. Dai, J. Grabowski, and Ø. Haugen, *Model-driven testing*. Springer, 2008.

14. IBM, Rational Unified Process, *Best Practices for Software Development Teams*, 2001.

15. Cohoon, J.P. (2002). *C++ program design: an introduction to programming and object-oriented design.* 清华大学出版社有限公司

16. J. Steele and N. Iliinsky, *Designing Data Visualizations: Representing Informational Relationships*. books.google.com, 2011.

17. J. J. Garrett, "A visual vocabulary for describing information architecture and interaction design," *Retrieved July*, 2002.

18. J. Gosling, *The Java language specification*. books.google.com, 2000.

19. M. Pilgrim, *HTML5: up and running*. books.google.com, 2010.

20. B. B. Bederson and J. D. Hollan, "Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics," in *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology*, New York, NY, USA, 1994, pp. 17–26.

21. B. B. Bederson, J. Meyer, and L. Good, "Jazz: an extensible zoomable user interface graphics toolkit in Java," *Proceedings of the 13th annual ACM ...*, 2000.

22. A. Druin, B. B. Bederson, and J. P. Hourcade, "Designing a digital library for young children," *... of the 1st ACM/IEEE-CS ...*, 2001.

23. M.-A. Storey, C. Best, J. Michaud, D. Rayside, M. Litoiu, and M. Musen, "SHriMP Views: An Interactive Environment for Information Visualization and Navigation," in *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2002, pp. 520–521.

24. Federico Rodríguez Bravo, "FVGAIA, Trabajo de grado: Framework para la Visualización de Grafos con Ayuda de Interfaces Aumentables", http://pegasus.javeriana.edu.co/~CIS1210TK05/index.html, 2012. [Último acceso: 13Mayo2015].

25. T. E. Hansen, J. P. Hourcade, M. Virbel, S. Patali, and T. Serra, "PyMT: A post-WIMP Multi-touch User Interface Toolkit," in *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, New York, NY, USA, 2009, pp. 17–24

26. G. Booch, J. Rumbaugh, I. Jacobson, J. S. Martínez, and ..., *El lenguaje unificado de modelado*. files.cecap49.webnode.es, 1999.

27. IEEE STD 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology", 1990

28. L. Zucconi, "Selecting a CASE Tool," *SIGSOFT Softw. Eng. Notes*, vol. 14, no. 2, pp. 42–44, Apr. 1989.

29. T. Kühne, “What is a Model,” in IBFI, 2005.

30. Villamor, C., Willis, D., Wroblewski, L.: Touch Gesture Reference Guide. (2010, April 15). LukeW Ideation + design: http://static.lukew.com/TouchGestureGuide.pdf [Último acceso: 13-Mayo-2015].

31. Karl E. Wiegers, First Things First: Prioritizing: http://www.processimpact.com/articles/prioritizing.html [Último acceso: 16-Mayo-2015].

32. “gnu.org.”: https://www.gnu.org/licenses/lgpl.html. [Último acceso: 16-Mayo-2015].

33. The Qt company, Qt Developers: http://qt-project.org/ [Último acceso: 16-Mayo-2015].

34. Zach Lieberman, Theodore Watson, Arturo Castro and OF community, open Frameworks an open source C++ toolkit: http://www.openframeworks.cc/ [Último acceso: 16-Mayo-2015].

35. Oliver Hamann, Eagle Mode, the most advanced Zoomable User Interface: http://eaglemode.sourceforge.net/ [Último acceso: 16-Mayo-2015].

36. GTK+ Team, GIMP Toolkit: http://www.gtk.org [Último acceso: 16-Mayo-2015].

37. Nuigroup, Touchlib library for creating multi-touch interaction surfaces: http://nuigroup.com/touchlib/ [Último acceso: 16-Mayo-2015].

38. Microsoft Corporation, Visual Studio: https://www.visualstudio.com/ [Último acceso: 16-Mayo-2015].

39. E. Montenegro, D. Rico, “Selección Framework”, Pontificia Universidad Javeriana: http://pegasus.javeriana.edu.co/~CIS1510AP01/documents/Selección%20Framework.xlsx [Último acceso: 23-Mayo-2015].

40. E. Montenegro, D. Rico, “Documentación ZoomTI++”, Pontificia Universidad Javeriana: http://pegasus.javeriana.edu.co/~CIS1510AP01/documentacion/annotated.html [Último acceso: 23-Mayo-2015].

41. E. Montenegro, D. Rico, “Especificación de Requerimientos”, Pontificia Universidad Javeriana: http://pegasus.javeriana.edu.co/~CIS1510AP01/documents/Especificaci%C3%B3n%20de%20Requerimientos.xlsx [Último acceso: 23-Mayo-2015].

42. E. Montenegro, D. Rico, "Priorización de requerimientos V2.0", Pontificia Universidad Javeriana: http://pegasus.javeriana.edu.co/~CIS1510AP01/documents/Priorización%20de%20requerimientos%20V2.0.pdf

43. "Qt | Cross-platform application & UI development framework," *Qt*. [Online]. Available: http://www.qt.io/. [Accessed: 25-May-2015].

44. Patrones de diseño, Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php. [Accessed: 25-May-2015].

45. "cplusplus.com" http://www.cplusplus.com/doc/tutorial/polymorphism/ [Último acceso: 25-Mayo-2015].

46. S. Kemp, *Ultimate Guide to Project Management*. Entrepreneur Press, 2005.

47. Jaime A. Pavlich, ZooMEnv-2.0 Concrete Syntax Requirements, presentación power point.

48. E. Montenegro, D. Rico, "SRS ZoomTI++ V2.0", Pontificia Universidad Javeriana: http://pegasus.javeriana.edu.co/~CIS1510AP01/documents/SRSZoomTIV2.0.pdf [Último acceso: 23-Mayo-2015].

49. E. Montenegro, D. Rico, "Manual de Usuario - Manual de Programador", Pontificia Universidad Javeriana: http://pegasus.javeriana.edu.co/~CIS1510AP01/documents/Manual%20de%20uso%20Zoomti++.pdf [Último acceso: 23-Mayo-2015].

50. "cmake.org" http://www.cmake.org/ [Último acceso: 17-Junio-2015].

51. "patriciogonzalezvivo/ofxBlackBox," *GitHub*.]. Disponible en: https://github.com/patriciogonzalezvivo/ofxBlackBox. [Último acceso: 18-Junio-2015].

IX - ANEXOS

Anexo 1: Glosario

Este anexo presenta los conceptos y las definiciones utilizados durante este trabajo de grado; incluyendo los documentos referenciados en los demás anexos y la memoria.

Anexo 2: SRS - Especificación de Requerimientos de Software

Este anexo presenta todos los pasos realizados para la escogencia de los requerimientos, incluye tres archivos: El primer documento incluye la información de importancia del *framework* a desarrollar, incluyendo usuarios potenciales, interfaces, entre otros. La hoja de cálculo relacionada incluye los requerimientos concretos de *ZoomTI++* y un sistema para medir el avance de la implementación del mismo. Finalmente el documento de priorización especifica los criterios tenidos en cuenta para darle prioridad a los requerimientos y descartar algunos.

Anexo 3: Selección Plataforma

Este anexo presenta de forma organizada todo el proceso realizado para la selección de la plataforma base, la cual fue utilizada para el desarrollo de *ZoomTI++*.

Incluye documento explicativo de las pruebas realizadas a cada una de las herramientas, proceso de eliminación de herramientas y las pruebas específicas realizadas a las herramientas finalistas. La hoja de cálculo en esta sección incluye los resultados puntuales de cada una de las pruebas.

Anexo 4: SAD - Documentación de Arquitectura de Software

Este anexo muestra la arquitectura con la que fue implementado ZoomTI++, presentando diferentes vistas del sistema.

Anexo 5: Manual de Uso de la Plataforma

Este anexo presenta un manual para el usuario de ZoomTI++, el documento relacionado muestra de qué manera se puede acceder a cada una de las funcionalidades de ZoomTI++ desde su ejecución.

Anexo 6: Repositorio ZoomTI++

Este anexo presenta un enlace al repositorio de código donde se encuentra almacenado *ZoomTI++*, el repositorio usado fue *Bitbucket*.