Facultad 5

Título: *Herramienta para generar máquinas de estados utilizadas en la lógica de las actividades de los laboratorios virtuales*

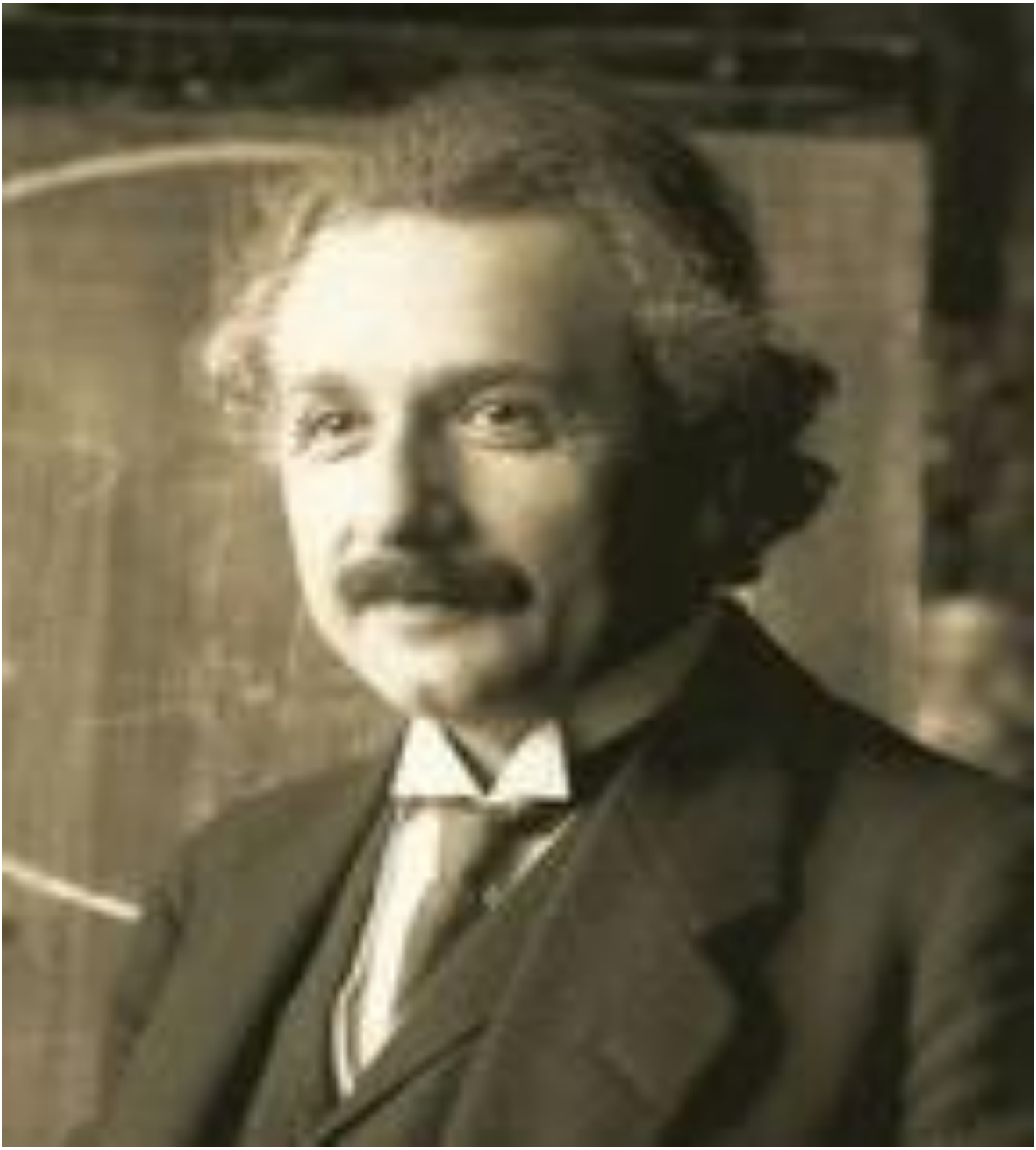
Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Luis Kayrumet Pérez Buigas

Tutor(es): Ing. Juan Miguel Rodríguez Sillero

Co-Tutor**:** Ing. Ángel Ulise Tabares González

La Habana, Junio 2014



"NUNCA CONSIDERES EL ESTUDIO COMO UNA OBLIGACIÓN, SINO COMO UNA OPORTUNIDAD PARA PENETRAR EN EL BELLO Y MARAVILLOSO MUNDO DEL SABER"

ALBERT EINSTEIN

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Juan Miguel Rodríguez Sillero

_____

Firma del Tutor

Ángel Ulise Tabares González

_____

Firma del Co-Tutor

Luis Kayrumet Pérez Buigas

_____

Firma del Autor

Datos de contacto

Tutor

Nombre y apellidos: Juan Miguel Rodríguez Sillero.

Especialidad: Ingeniero en Ciencias Informáticas, graduado en 2011

Correo Electrónico: jmsillero@uci.cu

Teléfono: 837-3376

Co-Tutor

Nombre y apellidos: Ángel Ulise Tabares Gonzales

Especialidad: Ingeniero en Ciencias Informáticas, graduado en 2012

Correo Electrónico: ulise@uci.cu

Teléfono: 835-8824

AGRADECIMIENTOS

A mis tutores por ayudarme y hacer posible que hoy esté aquí.

A mis padres que siempre han estado presente en los momentos más importantes de mi vida.

A mi hermana Suset por todo su apoyo y estar siempre regañándome cuando hago las cosas mal.

A mis otros hermanos y mis sobrinos por darme esos momentos de alegría que siempre hacen falta.

A toda mi familia que de una forma u otra siempre me apoyaron e hicieron posible este día.

A mi gordi por ayudarme en estos años y darme su apoyo de una forma u otra.

A mi amigo Joel que a pesar de la distancia seguimos siendo los mismos amigos de siempre.

A todos mis amigos del apto, Joaquín, Pedro, Osnier, Oscar, José Carlos, José Luis, Yusvel, Chang, Rafael y Pellicer, por estar siempre ahí para las discusiones de futbol, de pelota, y de cualquier otro tema. Por las fiestas que hicimos juntos, los juegos de dominó, nunca olvidare esos momentos.

A Elizabeth y Orlenis por estar siempre aguantando mis bromas.

A mis amigos del aula que aguantaron los 5 años de la carrera y ya pronto todos seremos ingenieros.

A todo el piquete del laboratorio 305 que todo este año lo hemos pasado junto, gracias por su ayuda.

A todas las personas que de una forma u otra posibilitaron este logro.

DEDICO ESTE TRABAJO A MI MAMÁ POR ESTAR SIEMPRE AHÍ PARA MÍ.

A MI PAPÁ POR APOYARME EN MIS DECISIONES Y CONFIAR EN MÍ.

A MI HERMANA QUE SIEMPRE ME HA AYUDADO Y COMPRENDIDO.

Resumen

Los laboratorios virtuales son medios educativos de gran utilidad, con ellos es posible simular el ambiente de un laboratorio tradicional y desarrollar una práctica de laboratorio sin correr los riesgos ni costear los gastos que conlleva una práctica de laboratorio tradicional. Algunos laboratorios virtuales utilizan una máquina de estados *para su desarrollo*, permitiendo que la implementación sea más organizada.

El presente trabajo tiene como objetivo el desarrollo de una herramienta que permita realizar un diseño en dos dimensiones (2D) de máquinas de estados para los laboratorios virtuales. La herramienta tendrá por nombre “Herramienta para generar máquinas de estados utilizadas en la lógica de las actividades de los laboratorios virtuales”. Para la creación de dicha herramienta se realizó una investigación de los elementos relacionados con las áreas de diseño de máquinas de estados y los fundamentos necesarios para lograr el diseño. Como resultado final se creó una herramienta que le permite al usuario realizar la tarea de diseño y luego exportar el diseño a código fuente C++ para su utilización en algún entorno de desarrollo integrado.

PALABRAS CLAVE:

Laboratorios virtuales, herramienta de diseño de máquina de estados

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con el avance de las Tecnologías de la Informática y las Comunicaciones prácticamente todos los aspectos cotidianos se han vistos reflejados en el fenómeno de la virtualización. Mediante la aplicación de la Realidad Virtual (RV), podemos hacer que el usuario interactúe con aplicaciones que simulan diferentes situaciones de la vida real. Los Laboratorios Virtuales (LVs) son un área de aplicación de la RV, que pretenden acercar al usuario al ambiente de un Laboratorio Tradicional (LT), los LVs tienen como objetivo facilitar la comprensión de conceptos científicos a través de la simulación de estos en un computador. Los experimentos se desarrollan paso a paso, como se haría en un LT, ya que se visualizan instrumentos y fenómenos mediante objetos dinámicos, imágenes o animaciones. (1)

La Universidad de las Ciencias Informáticas mediante el centro *VERTEX* ya ha desarrollado su propia versión de varios LVs, ejemplo de esto es el "Laboratorio Virtual Ensamblaje de una Computadora para la enseñanza de la Informática" (2) y se encuentra desarrollando nuevas versiones de otros. Durante el desarrollo de algunos de estos productos, una situación ha sido recurrente, y es el hecho de detectar la existencia de varios estados y transiciones en los comportamientos de estos. Una forma usual de solventar estas situaciones es la utilización del patrón máquina de estados a partir de un boceto del autómata finito determinista que cumple con estos estados y transiciones.

Una máquina de estados o autómata es un modelo de comportamiento de un sistema con entradas y salidas, donde un estado modela un comportamiento determinado. Al utilizarse este patrón se permite que la implementación sea más organizada y que el mantenimiento posterior sea más fácil. (3)

En estos momentos el centro no cuenta con ningún medio que permita automatizar este proceso de diseñar un autómata con las especificaciones que son necesarias como son el nombre de los estados, atributos de las clases y métodos de transición con sus parámetros. Actualmente cuando se realiza el diseño de un autómata, el desarrollador lo tiene que hacer de forma manual y escrita, el problema que trae consigo esto es que tras el diseño de un autómata, estos son propensos a no estar correctamente elaborados, ya que pueden existir errores humanos al elaborarlos que de pasar desapercibidos en una etapa más avanzada del desarrollo pueden inducir retrasos importantes que atentan contra el cumplimiento del cronograma. También al realizar el diseño de forma escrita no permite realizarlo con las especificaciones anteriormente mencionadas por lo que el proceso de implementación sería más extenso.

Cuando se procede a la implementación de este patrón el programador debe utilizar más tiempo para implementar desde el principio cada clase correspondiente a cada estado del diagrama, teniendo en cuenta la estructura que utiliza el patrón máquina de estados, esta tarea puede tornarse extensa y propensa a errores a medida que la cantidad de estados del autómata crece, lo que impide centrarse en la programación de las particularidades de cada ejercicio y subutiliza el tiempo en la resolución de tareas más comunes.

Teniendo en cuenta lo anteriormente planteado surge el siguiente **Problema de investigación:** ¿Cómo reducir los tiempos de desarrollo de los laboratorios virtuales que utilicen el patrón máquina de estados?

Se define como **Objeto de Estudio**: las herramientas para el diseño de autómatas derivándose como **Objetivo General**: Desarrollar una herramienta que permita generar código fuente, a partir del diseño de un autómata finito, para la lógica de los laboratorios virtuales, donde el **Campo de Acción** es: las herramientas para el diseño de autómatas con la lógica de un laboratorio virtual.

Para poder cumplir con el objetivo planteado es necesario realizar tareas investigativas que se definen a continuación:

1. Elaboración del marco teórico a través del estudio de la documentación existente sobre el tema en la actualidad.
2. Selección de las herramientas, lenguajes y tecnologías a utilizar para el desarrollo de la herramienta.
3. Elaboración de la propuesta de solución.
4. Análisis y diseño de la herramienta.
5. Implementación de la propuesta de solución.
6. Validación de la aplicación.

Para todo el proceso de investigación y elaboración de este trabajo se utilizaron los siguientes métodos científicos:

Teóricos:

- **Histórico – Lógico**: Método teórico mediante el cual se conocerá como ha sido la trayectoria histórica real, la evolución y desarrollo de los aspectos principales de las herramientas de diseño de autómatas.

- **Analítico – Sintético**: Este método teórico será utilizado en la investigación para buscar y analizar toda la documentación, permitiendo la extracción de las características más importantes de las herramientas de diseño de autómatas que puedan ser utilizadas en el desarrollo de laboratorios virtuales y llegar a conclusiones teóricas y prácticas del trabajo.

Empíricos:

- **Consulta de fuentes de información:** Método que permite tener acceso a la información disponible sobre el tema.

- **Entrevista:** Método empírico para obtener información amplia, abierta y directa de forma oral durante una conversación planificada sobre las herramientas de diseño de autómatas que puedan ser utilizadas en el desarrollo de laboratorios virtuales y su forma de utilización.

- **Observación:** Método empírico que permite conocer la realidad mediante la percepción directa de los objetos y fenómenos, es una manera de acceder a la información directa e inmediata sobre el proceso, fenómeno u objeto que está siendo investigado.

- **Consulta de especialistas:** Consulta a especialistas en las materias de diseño de autómatas, para conocer el funcionamiento y cómo hacer más fácil el entendimiento de la herramienta.

El presente trabajo tiene la siguiente estructura: resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas, glosario de términos y anexos.

Capítulo 1. Fundamentación teórica

- En este capítulo se hace un estudio de la bibliografía acerca del tema de la investigación, donde se abordan las diferentes herramientas que existen para el diseño de autómatas, principales características, ventajas y desventajas. Se presentan conceptos y definiciones que permiten una mejor comprensión del tema y ayudan al desarrollo de la solución.

Capítulo 2. Análisis y diseño

- En este capítulo se describe la solución teniendo en cuenta la selección de las herramientas y los lenguajes que se utilizan. Se especifican los requisitos funcionales y no funcionales de la aplicación, además de los patrones y estilos arquitectónicos utilizados, así como la descripción

de las historias de usuarios de la aplicación. Se muestran los diagramas de clases del análisis y las tarjetas CRC (Contenido, Responsabilidad y Colaboración) correspondientes a las clases.

Capítulo 3. Implementación y Validación de la propuesta

- Se realiza la implementación de la herramienta con los estándares de codificación del centro y se aplican las diferentes pruebas para validar la propuesta del desarrollo de la herramienta para generar máquinas de estados utilizadas en la lógica de las actividades de los laboratorios virtuales, mostrando los resultados de estas validaciones.

CAPÍTULO 1 FUNDAMENTACIÓN TEÓRICA

1. Introducción

En este capítulo se definen todos los conceptos referentes al problema de la investigación en cuestión, que permite dejar claro que es un Laboratorio Virtual y que es una máquina de estados finita y determinista, este capítulo también tiene la finalidad de recopilar la mayor cantidad de información posible acerca de las herramientas existentes para el diseño de máquinas de estados que se puedan utilizar para la lógica de los laboratorios virtuales.

Además se incluye un resumen de las características y ventajas de las herramientas y tecnologías a utilizar en el desarrollo de la herramienta que permita el diseño de autómatas utilizados en la lógica de los laboratorios virtuales.

1.1 Conceptos asociados al dominio del problema

Para obtener una mayor comprensión y desenvolvimiento de los contenidos que se abordarán en el presente y posteriores capítulos, se expondrán un grupo de conceptos identificados durante la investigación realizada. Estos conceptos son los siguientes:

1.1.1 Laboratorios Virtuales

Los laboratorios virtuales son entornos virtuales como ambientes de aprendizaje, que fomentan grandemente la integración de los usuarios a la práctica propiciando cada vez más simulaciones de fenómenos de la vida diaria.

Como concepto se conoce que son objetos digitales que, mediante el modelado de la realidad, permiten la simulación y la experimentación de fenómenos o situaciones de la vida real. Son un espacio electrónico de trabajo concebido para la colaboración y la experimentación a distancia con objeto de investigar o realizar otras actividades creativas y elaborar y difundir resultados mediante tecnologías de información y comunicación. (1)

1.1.2 UML

El Lenguaje de Modelado Unificado™ (UML) es la forma de los modelos mundiales no solo de la estructura de la aplicación, el comportamiento, y la arquitectura, sino de procesos de negocios y los datos de la estructura. UML, junto con la *Meta Object Facility* (MOF), también proporciona una base clave para la *Model-Driven Architecture* (OMG), que unifica en cada paso del desarrollo y la integración

del modelado de negocio, a través del modelado arquitectónico y de aplicación, con el desarrollo, el despliegue, el mantenimiento y la evolución. (4)

1.1.3 Máquina de estados

Una descripción excesivamente precisa de una máquina de estados es que es un grafo dirigido, que consiste en un conjunto de nodos y un conjunto correspondiente de funciones de transición. La máquina "corre" por responder a una serie de eventos. Cada evento está en el dominio de la función de transición que pertenece al nodo "actual", donde el rango de la función es un subconjunto de los nodos. La función devuelve el "siguiente" (quizás el mismo) nodo. Al menos uno de estos nodos deben ser un estado final. Cuando se alcanza un estado final, la máquina se detiene. (5)

1.1.4 Autómata finito determinista

Un autómata finito tiene un conjunto de estados, y su control "se mueve" de estado a estado en respuesta a "entradas" externas. Una de las distinciones fundamentales entre las clases de autómatas finitos es si ese control es "determinista", lo que significa que el autómata no puede estar en más de un estado en cualquier momento. (3)

A los efectos de este trabajo de define que una máquina de estado y un autómata representan el mismo concepto.

1.1.5 Estado

Un estado modela una situación durante la cual tiene alguna (por lo general implícita) condición invariante. El invariante puede representar una situación estática, como un objeto a la espera de que se produzca algún acontecimiento externo. Sin embargo, también puede modelar condiciones dinámicas, tales como el proceso de la realización de algún comportamiento. (6)

1.1.6 Transición

Una transición es una relación dirigida entre un estado origen y un estado destino. Puede ser parte de una transición compuesta, que tiene la máquina de estado de una configuración estado a otro, lo que representa la respuesta completa de la máquina de estados a una ocurrencia de un evento de un tipo particular. (6)

1.1.7 Patrón *State*

Es un patrón de diseño que se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo. Por ejemplo: una alarma puede tener diferentes estados, como desactivada, activada, en configuración. (7)

Clasificación del patrón: de comportamiento. (7)

Intención: Alterar el comportamiento de un objeto según el estado interno en que se encuentre. Así, el objeto aparentará cambiar de clase.

También conocido como: *Objects for States*.

Aplicabilidad: Se utiliza este patrón cuando:

- El comportamiento de un objeto depende de su estado y tiene que cambiar su comportamiento en tiempo de ejecución en función de ese estado.
- Se presentan muchos condicionales en el código, es posible que sea necesario aplicar este patrón.

Ventajas:

- Se localizan fácilmente las responsabilidades de los estados específicos. Esto facilita la ampliación de estados.
- Hace los cambios de estado explícitos puesto que en otros tipos de implementación los estados se cambian modificando valores en variables.
- Evita la utilización de estructuras condicionales.
- Impone una estructura sobre el código y hace más clara su intención.
- Hace explícitas las transiciones entre cuándo se tiene que ejecutar un comportamiento u otro.
- Permite a un objeto cambiar de clase en tiempo de ejecución.

1.1.8 Patrón *Command*

El patrón *Command* es un patrón comportamiento, que tiene como propósito encapsular peticiones en forma de objeto. También permite hacer colas de peticiones, llevar un registro de las peticiones realizadas y deshacer el efecto de las peticiones.

Es conocido también como acción o transacción. (7)

Para realizar la operación deshacer se debe almacenar el estado previo a la ejecución del comando como el objeto receptor, argumentos de la operación, valores del receptor que puedan cambiar tras ejecutar el comando.

Almacenar el estado anterior a la ejecución del comando permite hacer un deshacer.

Almacenar todos los estados anteriores permite hacer n deshacer.

1.2 Antecedentes de módulos de diseño 2D de autómatas

En este epígrafe se hace el estudio de las principales herramientas de diseño de autómatas que existen en la actualidad, con el fin de conocer sus principales características tanto en la interfaz de usuario como en las funcionalidades que brindan.

1.2.1 Cadifra UML Editor

Es una herramienta que permite dibujar diagramas UML con calidad profesional en Windows 7, Windows Vista o Windows XP.

Características (8)

- Posee una lista sin límite de deshacer y rehacer. Cada acción que modifique su diagrama UML se puede deshacer siempre, perfectamente y sin excepciones.
- Permite insertar diagramas UML en otros documentos (como los documentos de Microsoft Word). *Cadifra UML Editor* almacena el diagrama UML dentro del documento de la otra aplicación. Si se ejecuta Doble clic en el diagrama UML se abre *Cadifra UML Editor* para editar fácilmente.
- Da soporte completo de arrastrar y soltar. Permite pulsando la tecla ctrl crear una copia mientras arrastra (copia de movimiento).
- Muy fácil la edición de todos los textos. Basta con hacer doble clic en cualquier texto para editarlo.
- Posee un manejo inteligente de los conectores. Si se mueve, por ejemplo una clase, se ajustarán todas las asociaciones unidas, según sea necesario.
- Siempre se puede colocar cualquier segmento de un conector exactamente donde usted lo quiere. *Cadifra UML Editor* soporta dibujo de conectores en forma de árbol de cualquier complejidad.
- Tiene un editor de estilo para definir las fuentes de los textos de las familias de los elementos de UML. El editor de estilo utiliza un modelo de herencia. Puede establecer la fuente predeterminada de todo el diagrama UML y establecer una fuente primordial para un tipo específico de elemento UML. Los cambios se aplican a todos los elementos existentes en el diagrama.

- Tiene alta precisión de gráficos: Líneas y flechas terminan exactamente en cajas. No hay necesidad de recurrir a una herramienta de dibujo genérico sin el conocimiento incorporado en UML de primera mano.
- *Cadifra UML Editor* es muy rápido, fácil e intuitivo de usar. El usuario pasará menos tiempo capturando sus ideas con respecto a una herramienta UML de gran tamaño o una herramienta de dibujo.

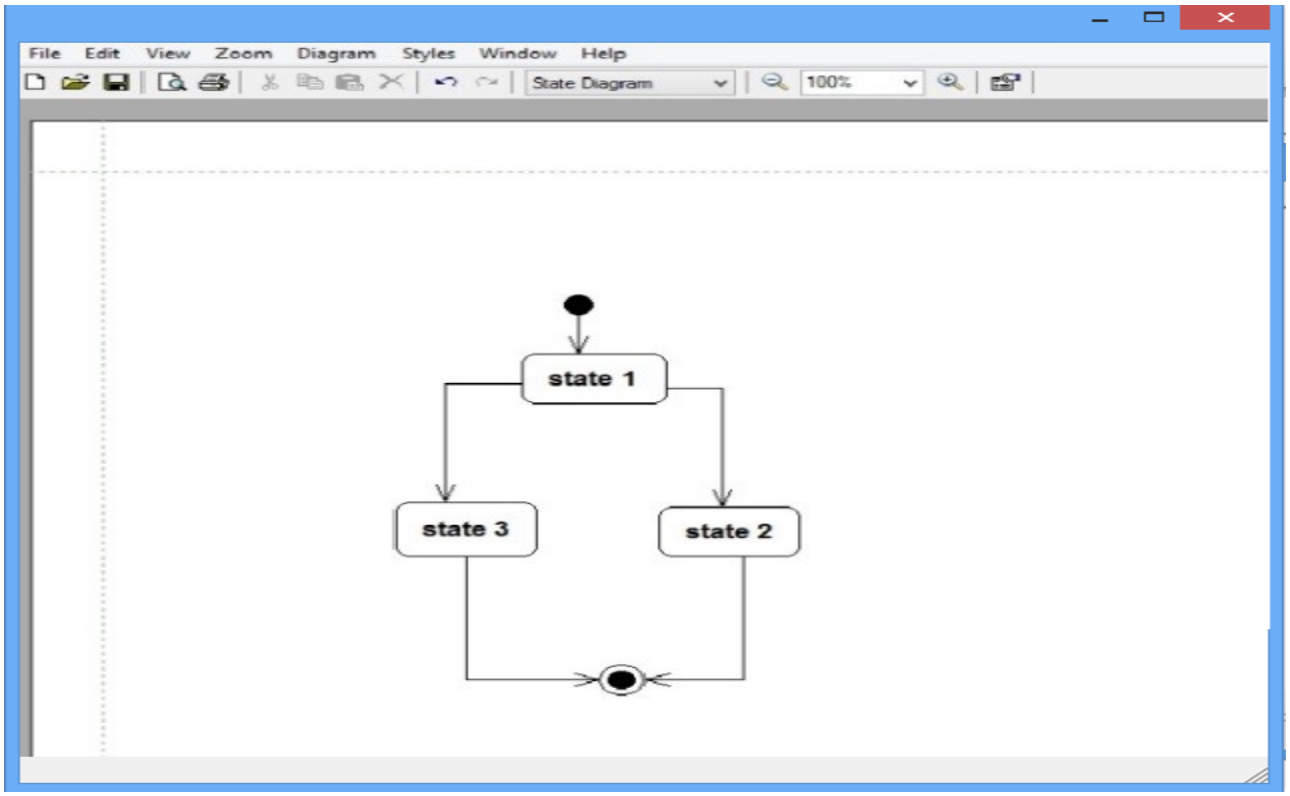


Figura 1 *Cadifra UML Editor*

Esta herramienta tiene varios inconvenientes entre los que se encuentra que no permite exportar el diagrama a código fuente para su utilización, además de que no se puede probar el funcionamiento de la máquina de estados.

1.2.2 Automata editor

Automata Editor es un editor vectorial para dibujar autómatas finitos. El editor permite exportar a diversos formatos, así como los formatos de mapa de bits (por ejemplo BMP, JPEG). (9)

El programa es una herramienta básica que permite trabajar con los autómatas finitos, exportar la tabla de transición y simular el trabajo de los autómatas finitos con posibilidad de exportar también los pasos de la simulación.

Características

- Posee una lista infinita de deshacer y rehacer.
- Permite personalizar cada uno de los estados (ejemplo color de relleno, color y tipo de borde).
- Permite simular el autómata diseñado y probar su correcto funcionamiento.
- Permite además generar máquinas de estados para reconocer una cadena de texto.

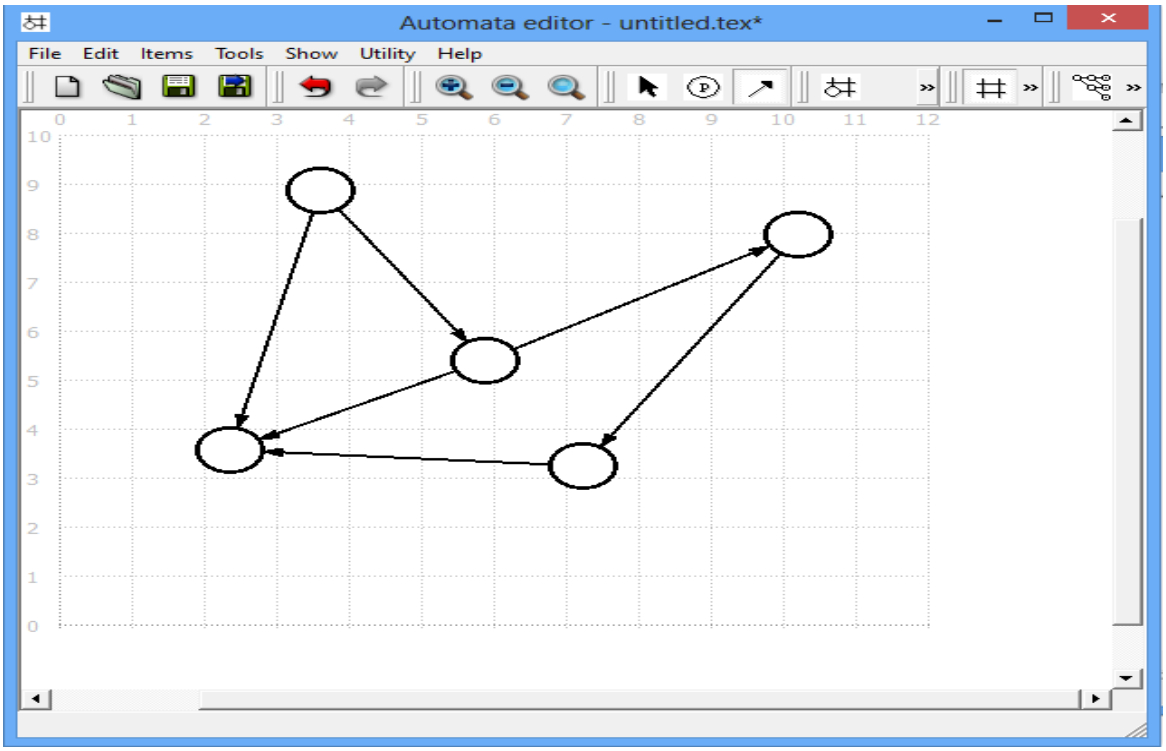


Figura 2 *Automata editor*

Esta herramienta tiene como inconveniente que a los estados no se le pueden agregar información necesaria como atributos y funciones y además no permite exportar el autómata a código fuente.

1.2.3 *Altova UModel* ® 2014

Esta herramienta es el punto de partida para el desarrollo de software con éxito. Permite diseñar de forma visual modelos de aplicaciones en UML y generar código Java, C # o *Visual Basic*.NET y la documentación del proyecto. Permite hacer ingeniería inversa de los programas existentes en diagramas de arquitectura de software en UML 2 y luego afinar sus diseños y completar la regeneración de código. *UModel* es la herramienta UML que permite el diseño de software de forma práctica y visual para cualquier proyecto. (10)

Características (11)

- Tiene una barra de herramientas de acceso rápido a los elementos del diagrama de máquina de estados.
- Las transiciones en los diagramas de máquina de estados pueden referirse a operaciones en las clases.
- Genera código Java, C # o VB. NET a partir de diagramas de máquina de estados.
- Tiene estilos en cascada de colores, tipos de letra y tamaño de la línea.
- Posee una rejilla de alineación personalizable.
- Los elementos pueden ser asignados a las capas del diagrama y ver u ocultar de forma selectiva.
- Posee una lista sin límite de deshacer y rehacer.

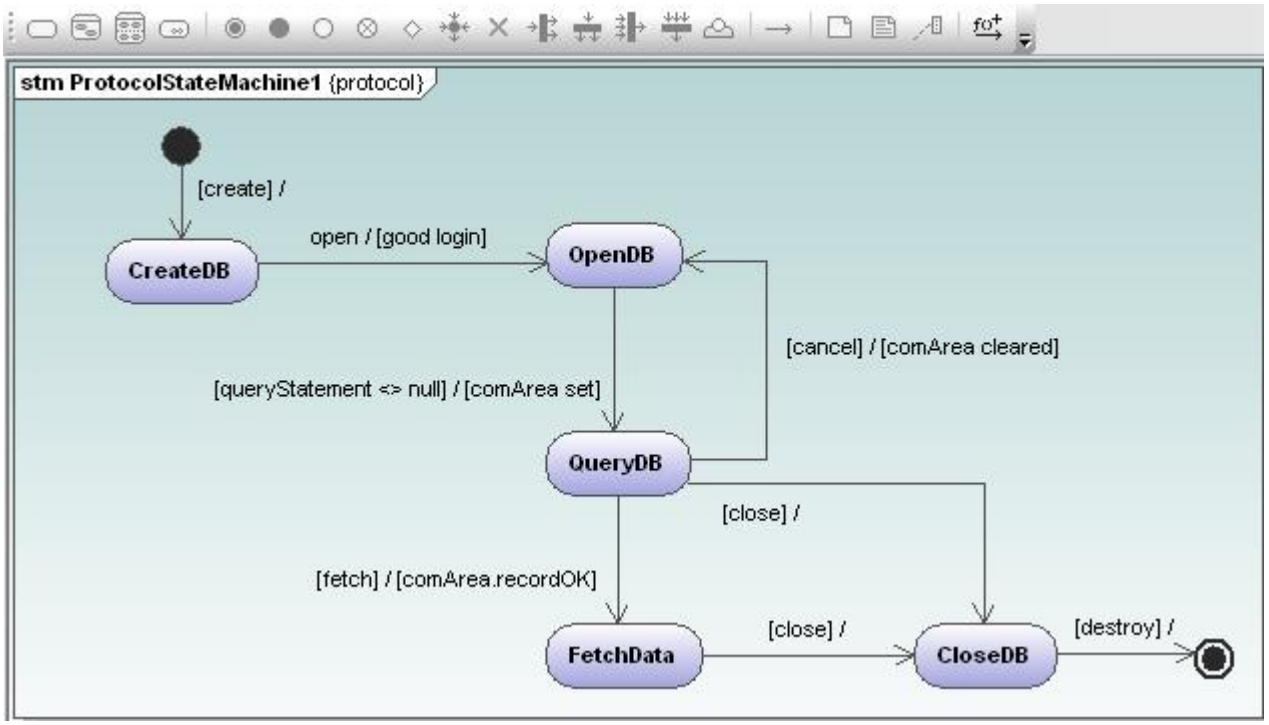


Figura 3 *Altova UModel* ® 2014

Esta herramienta tiene como desventaja que no permite simular el autómata diseñado por un usuario para conocer si funciona correctamente, además de que es privativa.

1.2.4 *Enterprise Architect* 7.0

Enterprise Architect es una herramienta potente de modelado UML 2.0. La aplicación es compatible con los 13 diagramas de UML 2.0. Posee una amplia gama de características para el desarrollo de

software, modelado de base de datos, requisitos, diseño de interfaz y la generación de documentos informes en formato HTML. (12)

Características: (12)

- Permite realizar el diseño y la construcción de UML.
- Permite crear casos de usos y modelos lógicos, dinámicos y físicos.
- Es intuitivo y fácil de usar.
- Da soporte para *ActionScript* 2.0, Java, C#, C++, VB.Net, *Delphi*, *Visual Basic*, *Python* y PHP.
- Permite importar y exportar en formato XML.
- Posee en corrector ortográfico.
- Es una herramienta que permite probar el autómata diseñado.

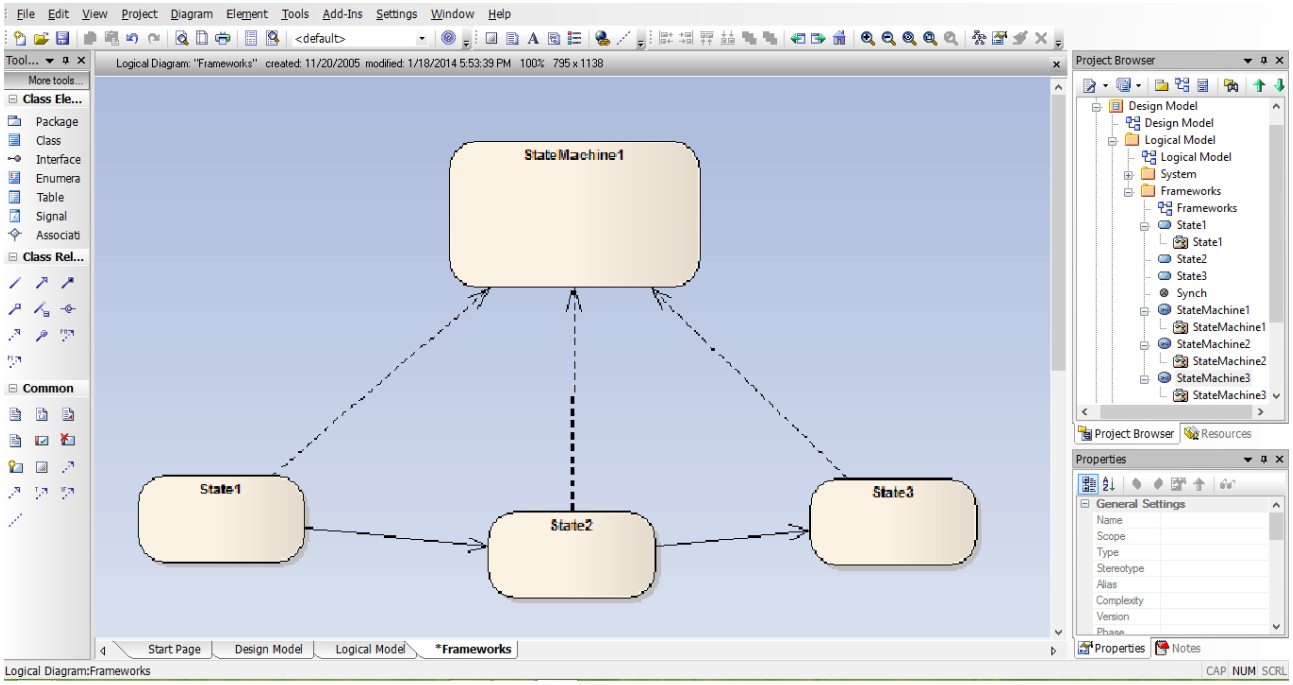


Figura 4 *Enterprise Architect* 7.0

Esta herramienta tiene como inconveniente que debido a que posee una gran cantidad de funcionalidades (12) puede provocar un mayor tiempo de asimilación, por lo que el problema de la investigación no se resolvería. Otra desventaja que posee es que no se le pueden agregar nuevas funcionalidades, ya que se está desarrollando una herramienta que va a ser integrada a la base técnica

de los LVs y en el futuro pudieran desearse otras funcionalidades. Además que esta aplicación es privativa y no es multiplataforma.

1.2.5 *Visual Paradim*

Visual Paradigm for UML Professional Edition (poco conocido como VP-UML PE) es una herramienta de casos de uso. Es compatible con todo lo necesario en la identificación y la organización de los requisitos del sistema a través del análisis de casos de uso. (13)

Características (14)

- Soporta todos los diagramas UML (clase, caso de uso, colaboración, secuencia, estado, actividad, componente, despliegue).
- Interfaz de usuario *Dockable*.
- Permite importar proyecto en XML
- Permite exportar proyecto en formato XML.
- Edición directa de los objetos VP UML OLE en otras aplicaciones.
- Permite generar documentación en formato HTML.
- Permite generar documentación en formato PDF.
- Da la posibilidad de exportar diagrama como imagen (JPG, PNG y SVG).
- Soporta multilingüe.

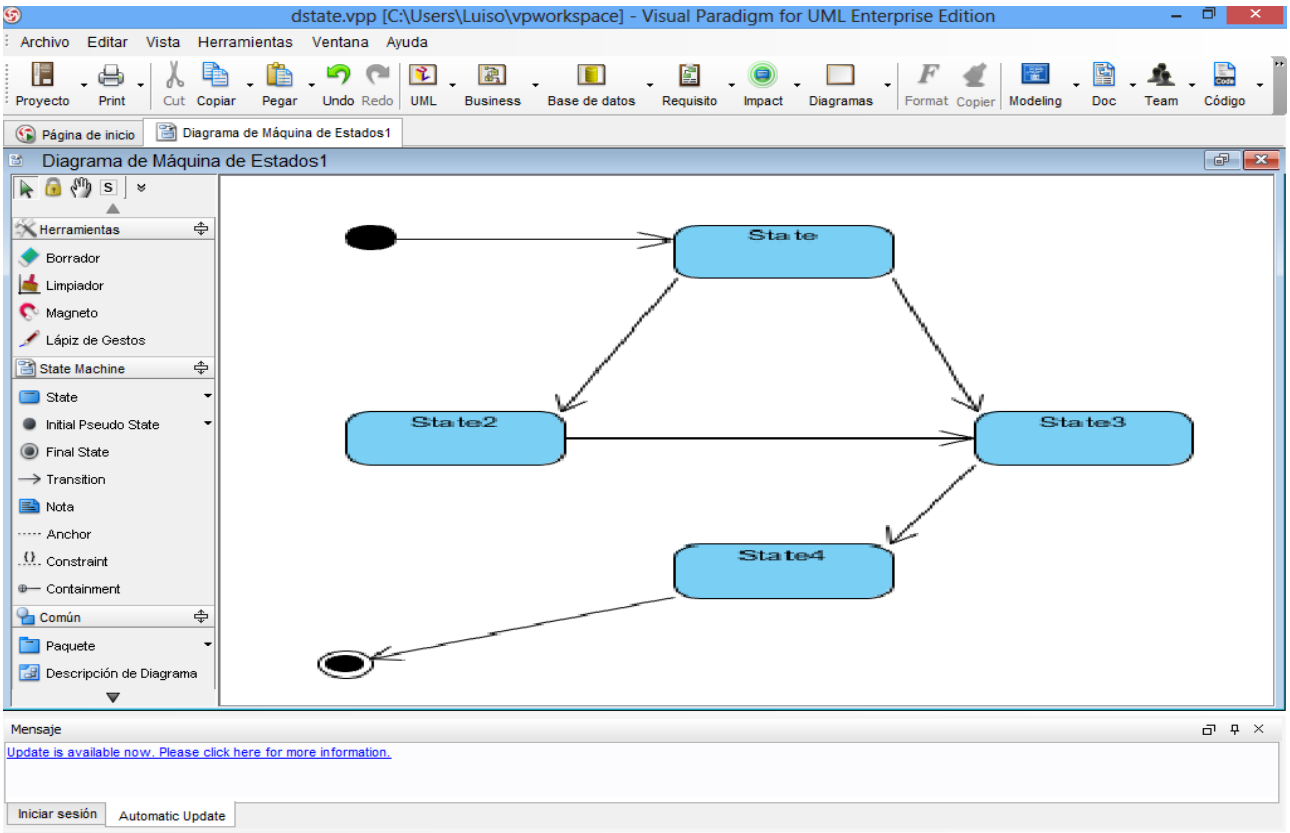


Figura 5 *Visual Paradim*

Esta herramienta no permite simular el autómata diseñado por un usuario y por lo tanto no se puede saber si la máquina de estados funciona correctamente.

1.2.6 *IBM Rational Rose Enterprise Edition*

Es una herramienta CASE (*Computer Aided Software Engineering*), traducido al español como Ingeniería Asistida por Computadora, desarrollada por *Rational Corporation* basada en UML, que permite crear los diagramas que se generan durante el proceso de ingeniería en el desarrollo del software.

Rational Rose Enterprise es el producto más completo de la familia Rational Rose. Todos los productos *Rational Rose* incluyen soporte UML. (15)

Rational Rose Enterprise es una elección para el ambiente de modelado que soporta la generación de código a partir de modelos. Como todos los demás productos *Rational Rose*, proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad rápidamente. (16)

Características (17)

- Posee características de control por separado de componentes que permite una administración granular y el uso de modelos.
- Permite la generación de código Ada, C++, CORBA, Java y Visual Basic con capacidad de sincronización modelo-código configurables.
- Posee la capacidad de análisis de la calidad de código.
- Da la posibilidad de modelar en UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
- Tiene la capacidad de crear definiciones de tipo de documento XML para el uso en la aplicación.
- Permite la integración con otras herramientas de desarrollo de *Rational Rose*.
- Permite la publicación web y generación de informes para optimizar la comunicación dentro del equipo.

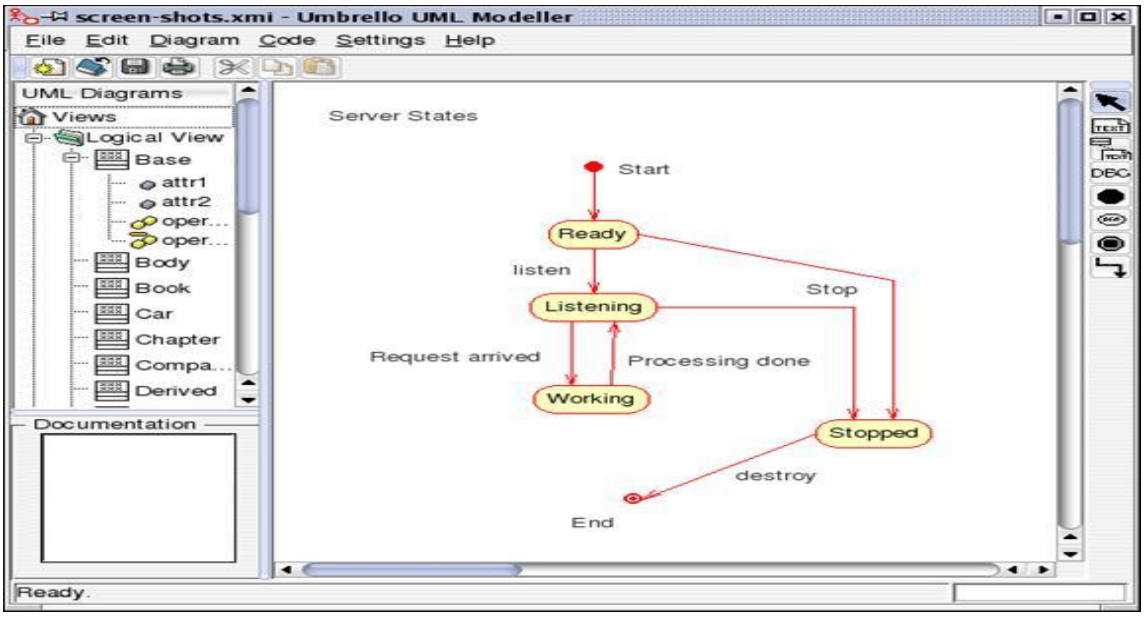


Figura 6 *IBM Rational Rose Enterprise Edition*

Esta herramienta tiene como desventaja que no permite simular el funcionamiento de la máquina de estados y por tanto no se puede asegurar su correcto funcionamiento.

1.2.7 Herramienta de configuración de estados y animaciones de un agente virtual autónomo para videojuegos 2D (HCEA)

Es una herramienta creada en la Universidad de la Ciencias Informáticas, que permite definir una configuración para el comportamiento inteligente de agentes autónomos en videojuegos 2D a través de

una máquina de estados. Permite gestionar agentes, escenas y estados. Además permite guardar y cargar las configuraciones realizadas en XML. (18)

Es herramienta tiene como inconveniente que no permite exportar la máquina de estados a código fuente C++. También presenta otro inconveniente, no permite probar el funcionamiento del autómata.

1.2.8 Comparación entre las herramientas existentes

Luego de realizar un estudio de las principales características de cada herramienta (Ver en anexos tabla comparativa), se pudo identificar algunas características comunes, como son:

- Exportar e importar en formato XML.
- Lista infinita de deshacer y rehacer.
- Guardar el diseño como imagen.
- Exportar a código el diagrama diseñado.
- Algunas de estas aplicaciones son multiplataforma y solo dos permitían la posibilidad de probar el funcionamiento de la máquina de estados diseñada.

Después de realizar el estudio de las soluciones existentes expuestas anteriormente la más cercana a la solución del problema planteado es la herramienta *Enterprise Architect* 7.0. Dicha aplicación ayuda a realizar personalizaciones sobre el diagrama, efectuando operaciones básicas de gestión tales como: adicionar, modificar y eliminar de estados y transiciones. A pesar de todas las ventajas que presenta no se puede utilizar por los inconvenientes descritos anteriormente en la sección 1.2.4.

Consideraciones del capítulo

En este capítulo fueron abordados diferentes conceptos que son necesarios para la comprensión del problema y así como el estudio de las diferentes de las herramientas existente para el diseño de máquinas de estados, especificando algunos de los aspectos más importantes relacionados con las mismas, los cuales propiciaron esclarecer las dudas existentes sobre el tema en cuestión y a la vez se comprobó que es necesario crear en herramienta para dar solución al problema de este trabajo.

A partir del estudio de las herramientas existente se adoptó como posible interfaz gráfica para la que se quiere desarrollar una interfaz gráfica similar a la que presenta la herramienta *Automata Editor*, agregándole las funcionalidades que brinda la herramienta *Enterprise Architect* 7.0 para poder darle mayor información a los estados y probar el funcionamiento de las máquinas de estados.

CAPÍTULO 2 ANÁLISIS Y DISEÑO

2. Introducción

En el presente capítulo se observarán los elementos relacionados con la solución desarrollada, teniendo en cuenta varios aspectos como la metodología, el lenguaje de programación y el entorno de desarrollo integrado que se utilizó. Además se hace mención de las características generales que tiene la herramienta que se desarrolló, también se definen los requerimientos del sistema, tanto los funcionales como los no funcionales. Otro aspecto que se aborda es la arquitectura que se seleccionó para la implementación de la herramienta, los patrones de diseño de programación utilizados entre otros aspectos importantes.

2.1 Metodologías y herramientas a utilizar para llevar a cabo la solución

Para lograr la correcta implementación de la herramienta se seleccionó el siguiente ambiente de desarrollo:

- ✓ Como lenguaje de programación se utilizará C++, que es un lenguaje orientado a objetos. Se encuentra entre los más usados para desarrollar aplicaciones gráficas, por ser el lenguaje que más rápido se ejecuta. C++ es un lenguaje de propósito general, o sea, que con él se pueden realizar muchas aplicaciones, desde sistemas operativos y compiladores hasta aplicaciones de bases de datos y procesadores de texto, juegos, entre otros. (19)
- ✓ Otro leguaje que se utilizara es XML, siglas en inglés de *eXtensible Markup Language* ('lenguaje de marcas extensible'), es un lenguaje utilizado para almacenar datos de forma legible. Permite definir la gramática de lenguajes específicos (de la misma forma que HTML) para estructurar documentos grandes. (20)
- ✓ Se usó seleccionó como entorno de desarrollo integrado *Qt Creator* debido a que este es un software libre, posee la ventaja de ser multiplataforma y tiene un avanzado editor de código C++, que es el lenguaje de programación seleccionado para el desarrollo del módulo, además cuenta con una interfaz gráfica de usuario (GUI) integrada y diseñador de formularios. (21)
- ✓ Como metodología de desarrollo de software se escogió Programación Extrema (XP), porque constituye una solución a medida en el entorno de trabajo donde el equipo de desarrollo es

pequeño y con plazos reducidos, proporcionando una elevada simplificación sin renunciar a las buenas prácticas para asegurar la calidad del software. (22)

2.2 Requisitos funcionales

Los requisitos funcionales son características requerida del sistema que expresa una capacidad de acción del mismo, una funcionalidad; generalmente expresada en una declaración en forma verbal. Los requisitos funcionales están enfocados a las funcionalidades y el comportamiento interno de la herramienta. (23) Los requisitos funcionales de la herramienta son:

- RF 1. Insertar estados.
- RF 2. Modificar estados.
- RF 3. Eliminar estados.
- RF 4. Insertar conexiones.
- RF 5. Modificar conexiones.
- RF 6. Eliminar conexiones.
- RF 7. Seleccionar uno o varios elementos en la escena.
- RF 8. Mover los elementos en la escena.
- RF 9. Deshacer y rehacer.
- RF 10. Gestionar características de los estados.
- RF 11. Realizar zoom en la escena.
- RF 12. Guardar imagen del diseño.
- RF 13. Guardar el diseño de la máquina de estados en un archivo en formato XML.
- RF 14. Cargar el diseño de la máquina de estados desde un archivo en formato XML.
- RF 15. Exportar a código C++.
- RF 16. Probar el funcionamiento de un autómata.

2.3 Requisitos no funcionales

Los requisitos no funcionales son aquellos que especifican los criterios que pueden usarse para juzgar la operación del sistema, no su comportamiento específico (23). Los requisitos no funcionales presentes en el módulo son los siguientes:

1. **Hardware**
   a) Procesador Pentium 4 a 2.7 GHz o superior.
   b) Memoria RAM de 512 MB de capacidad.
2. **Usabilidad**
   a) El sistema deberá ser multiplataforma, es decir deberá correr sobre los sistemas operativos Windows y GNU/Linux.
3. **Requisitos no funcionales de Restricciones en el Diseño e Implementación**
   a) Lenguaje de programación C++.
   b) IDE de desarrollo: Qt Creator.
   c) Bibliotecas de clases: Qt.

2.4 Fase de Exploración

La primera fase de la metodología de desarrollo XP es exploración, en esta fase los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Asimismo el equipo de trabajo se familiariza con la tecnología, las herramientas y prácticas a utilizar en el proyecto. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología. En esta fase, el cliente define lo que necesita mediante la redacción de sencillas “historias de usuarios”. Los programadores estiman los tiempos de desarrollo en base a esta información. Debe quedar claro que las estimaciones realizadas en esta fase son primarias, ya que estarán basadas en datos de muy alto nivel, y podrían variar cuando se analicen más en detalle en cada iteración. (24)

2.4.1 Historias de Usuarios

Las Historias de Usuarios (HU) son unos de los artefactos más importantes que genera la metodología de desarrollo ágil XP. Estas tienen el mismo propósito que los casos de uso y son escritas por el propio cliente, tal y como ven ellos las necesidades del sistema, por tanto son descripciones cortas y escritas en el lenguaje del usuario, sin terminología técnica, se realiza una por cada funcionalidad del sistema, se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos, reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar

lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre 1 y 3 semanas, estas permiten responder rápidamente a los requerimientos cambiantes. (24)

2.4.2 Relación de las Historias de Usuarios

Como parte del proceso de trabajo dentro de la fase de exploración se muestran a continuación las HU:

Tabla 1 Historia de usuario 1

Historia de Usuario	
Número: 1	**Usuario:** Cliente
Nombre: Insertar estados	
Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 1/4	**Iteración asignada:** 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta permitirá insertar estados en la escena	
Observaciones:	

Tabla 2 Historia de usuario 2

Historia de Usuario	
Número: 2	**Usuario:** Cliente
Nombre: Modificar estados	
Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 2/4	**Iteración asignada:** 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta permitirá modificar estados de la escena	
Observaciones:	

Tabla 3 Historia de usuario 3

Historia de Usuario	
Número: 3	**Usuario:** Cliente
Nombre: Eliminar estados	

Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 1/4	**Iteración asignada:** 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta permitirá eliminar estados de la escena	
Observaciones:	

Tabla 4 Historia de usuario 4

Historia de Usuario	
Número: 4	**Usuario:** Cliente
Nombre: Insertar conexiones	
Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1/4	**Iteración asignada:** 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta permitirá agregar conexiones entre los estados previamente insertados	
Observaciones:	

Tabla 5 Historia de usuario 5

Historia de Usuario	
Número: 5	**Usuario:** Cliente
Nombre: Modificar conexiones.	
Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 2/4	**Iteración asignada:** 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta permitirá modificar conexiones existentes entre los estados	
Observaciones:	

Tabla 6 Historia de usuario 6

Historia de Usuario	
Número: 6	**Usuario:** Cliente
Nombre: Eliminar conexiones.	

Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1/4	**Iteración asignada:** 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta permitirá eliminar conexiones existentes entre los estados	
Observaciones:	

Tabla 7 Historia de usuario 7

Historia de Usuario	
Número: 7	**Usuario:** Cliente
Nombre: Seleccionar uno o varios elementos en la escena	
Prioridad del negocio: Media	**Riesgo en desarrollo**: Medio
Puntos estimados: 1	**Iteración asignada**: 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe permitir seleccionar uno o varios elementos (estados, conexiones) en la escena	
Observaciones:	

Tabla 8 Historia de usuario 8

Historia de Usuario	
Número: 8	**Usuario**: Cliente
Nombre: Mover los elementos en la escena	
Prioridad del negocio: Media	**Riesgo en desarrollo**: Medio
Puntos estimados: 1	**Iteración asignada**: 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe permitir mover los elementos que se seleccionen en la escena	
Observaciones:	

Tabla 9 Historia de usuario 9

Historia de Usuario	
Número: 9	**Usuario**: Cliente
Nombre: Deshacer y rehacer	

Prioridad del negocio: Media	**Riesgo en desarrollo**: Alto
Puntos estimados: 1	**Iteración asignada**: 1
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe permitir deshacer y rehacer las acciones de insertar, mover y eliminar elementos de la escena	
Observaciones:	

Tabla 10 Historia de usuario 10

Historia de Usuario	
Número: 10	**Usuario:** Cliente
Nombre: Gestionar especificaciones de los estados	
Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta permitirá agregar, modificar y eliminar especificaciones de los estados (atributos, funciones, parámetros de las funciones)	
Observaciones:	

Tabla 11 Historia de usuario 11

Historia de Usuario	
Número: 11	**Usuario:** Cliente
Nombre: Realizar zoom en la escena	
Prioridad del negocio: Baja	**Riesgo en desarrollo:** Medio
Puntos estimados: 1	**Iteración asignada:** 2
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: En la escena se debe permitir alejar o acercar la escena para una mejor visualización del trabajo	
Observaciones:	

Tabla 12 Historia de usuario 12

Historia de Usuario	
Número: 12	**Usuario**: Cliente

Nombre: Guardar imagen del diseño en formato jpg	
Prioridad del negocio: Media	**Riesgo en desarrollo**: Medio
Puntos estimados: 1	**Iteración asignada**: 2
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe permitir guardar una imagen del diseño realizado una vez que el usuario haya terminado	
Observaciones:	

Tabla 13 Historia de usuario 13

Historia de Usuario	
Número: 13	**Usuario:** Cliente
Nombre: Guardar el diseño de la máquina de estados en un archivo en formato XML	
Prioridad del negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 2	**Iteración asignada:** 3
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe tener un mecanismo que permita seleccionar cualquier directorio del sistema y guardar en un archivo el diseño de la máquina de estados realizada	
Observaciones:	

Tabla 14 Historia de usuario 14

Historia de Usuario	
Número: 14	**Usuario:** Cliente
Nombre: Cargar el diseño de la máquina de estados desde un archivo un formato XML	
Prioridad del negocio: Alta	**Riesgo en desarrollo**: Alto
Puntos estimados: 1	**Iteración asignada**: 3
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe tener un mecanismo que permita seleccionar un fichero en cualquier directorio del sistema y poder cargar el archivo en formato XML que contiene el diseño de la máquina de estados creado por la herramienta anteriormente	
Observaciones:	

Tabla 15 Historia de usuario 15

Historia de Usuario	
Número: 15	**Usuario:** Cliente
Nombre: Exportar a código C++	
Prioridad del negocio: Alta	**Riesgo en desarrollo:** Muy Alto
Puntos estimados: 2	**Iteración asignada:** 3
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe ser capaz de exportar la máquina de estados a código C++ luego de ser creada por el usuario	
Observaciones:	

Tabla 16 Historia de usuario 16

Historia de Usuario	
Número: 16	**Usuario:** Cliente
Nombre: Probar el funcionamiento de un autómata	
Prioridad del negocio: Alta	**Riesgo en desarrollo**: Muy Alto
Puntos estimados: 2	**Iteración asignada:** 4
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: La herramienta debe permitir probar el funcionamiento de la máquina de estados	
Observaciones:	

2.5 Fase de Planificación

La fase de planificación lleva a cabo una estimación del esfuerzo que costará implementar cada una de las Historias de Usuarios. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Como unidad de medida se utilizará el punto de estimación, considerando que una semana de trabajo sin interrupciones es equivalente a un punto.

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. (24)

Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según el alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación. (24)

2.5.1 Estimación de esfuerzos

Para lograr un desarrollo eficiente y satisfactorio, se realizó una estimación de esfuerzos para cada una de las Historias de Usuarios identificadas en el proceso de planificación, llegando a los resultados que se muestran a continuación.

Tabla 17 Estimación de Esfuerzos

Historia de Usuario	Puntos de Estimación
Insertar estados	¼ semana
Modificar estados	¾ semana
Eliminar estados	¼ semana
Seleccionar uno o varios elementos en la escena	1 semana
Insertar conexiones	¼ semana
Modificar conexiones	¾ semana
Eliminar conexiones	¼ semana
Mover los elementos en la escena	1 semana

Deshacer y rehacer	1 semana
Gestionar características de los estados	2 semanas
Realizar zoom en la escena	1 semana
Guardar imagen del diseño	1 semana
Guardar el diseño de la máquina de estados en un archivo en formato XML	2 semanas
Cargar el diseño de la máquina de estados desde un archivo en formato XML	1 semana
Exportar a código C++	2 semanas
Probar el funcionamiento de un autómata	2 semanas

2.5.2 Plan de Iteraciones

En esta fase se incluyen varias iteraciones donde el plan de entrega de cada iteración no excederá las 5 semanas, después de identificadas y descritas las HU y estimado el esfuerzo para la realización de cada una de ellas, se procede a la planificación de la etapa de implementación de la herramienta. Es éste se especifica cuáles son las HU que se implementarán en cada iteración y se determinan las posibles fechas de entrega. Después de haber definido y estimado las HU se tomó la decisión de realizar 4 iteraciones, las cuales se detallan a continuación.

Iteración # 1

La primera iteración tiene como objetivo la implementación de las HU #1, #2, #3, #4, #5, #6, #7, #8 y #9 con prioridad para el cliente de Alta las 6 primeras y de Media las 3 últimas. En esta fase las HU insertar, modificar y eliminar estados es la base para implementar las demás iteraciones. Se dispone de 4 semanas para implementar todas las tareas. Se obtiene como resultado una primera versión del sistema propuesto la cual será mostrada al cliente para realizar la retroalimentación con el equipo de desarrollo, para luego pasar a la siguiente iteración.

Iteración # 2

En esta iteración se dará cumplimiento a las HU #10, #11 y #12. Se cuenta con 4 semanas para llevar a cabo la implementación de esta iteración. Al finalizar se obtendrá una segunda versión del sistema

propuesto y se le hará llegar al cliente la iteración anterior junto con la presente para la aprobación o cambios pertinentes con el cliente.

Iteración # 3

La tercera iteración corresponde a las HU #13, #14 y #15, en esta iteración se desarrollan las tareas Guardar, Cargar y Exportar un proyecto seleccionado. Como resultado se dará la posibilidad de exportar la máquina de estados a código fuente C++ a un directorio específico. Luego de ser aprobada por el cliente esta iteración se pasará a la última iteración.

Iteración # 4

La iteración # 4 y última corresponde a la HU #16, la cual tiene como objetivo probar el funcionamiento de la máquina de estados. En esta iteración se completará el 100% de las funcionalidades del sistema propuesto para ser probada por el cliente en su totalidad.

Las iteraciones a desarrollar presentan un nivel medio y alto de complejidad para el desarrollo con alta prioridad para el cliente. Se dispondrán de 17 semanas para desarrollar todas las iteraciones propuestas.

2.5.3 Plan de duración de las iteraciones

Para lograr una mayor organización del trabajo se crea un plan de duración de las iteraciones; el mismo tiene como objetivo mostrar la duración de cada iteración así como el orden en que serán implementadas las historias de usuarios en cada una de ellas.

Tabla 18 Plan de duración de Iteraciones

No Iteración	Historia de Usuario	Duración Total de Iteraciones
Iteración # 1	Insertar estados	5 semana
	Modificar estados	
	Eliminar estados	
	Seleccionar uno o varios elementos en la escena	
	Insertar conexiones	
	Modificar conexiones	
	Eliminar conexiones	

	Mover los elementos en la escena	
	Deshacer y rehacer	
Iteración # 2	Gestionar características de los estados	4 semana
	Permitir el zoom en la escena	
	Guardar imagen del diseño	
Iteración # 3	Guardar el diseño de la máquina de estados en un archivo en formato XML	5 semana
	Cargar el diseño de la máquina de estados desde un archivo en formato XML	
	Exportar a código C++	
Iteración # 4	Probar el funcionamiento de un autómata	2 semana

2.5.4 Plan de Entrega

A continuación se muestra el plan de entregas desarrollado para dar solución al problema planteado. Para desarrollar el mismo se tuvo en cuenta los puntos de estimación para obtener un resultado final.

Tabla 19 Plan de entrega de Iteraciones

No Iteración	Duración	Fecha Inicio	Fecha Final
Iteración # 1	5 semanas	27/1/2014	28/2/2014
Iteración # 2	4 semanas	3/3/2014	28/3/2014
Iteración # 3	5 semanas	31/3/2014	10/5/2014
Iteración # 4	2 semanas	12/4/2014	30/5/2014

2.6 Propuesta de la solución

Se propone desarrollar una herramienta que permita diseñar máquinas de estados, donde se gestionen los estados, los atributos, funciones y las transiciones referentes a cada uno de los estados. Debe permitir también probar el funcionamiento del autómata, además de guardar y cargar el diseño para su posterior utilización y deberá permitir además exportar el diseño a código fuente C++, para su utilización en la lógica de los LVs.

La herramienta contará con un área de trabajo, donde se podrá diseñar la máquina de estados, también contará con una barra de herramientas donde se encontrarán las principales acciones que se pueden realizar sobre el área de trabajo, a la derecha se colocará un menú que tendrá las principales características de los estados y las transiciones.

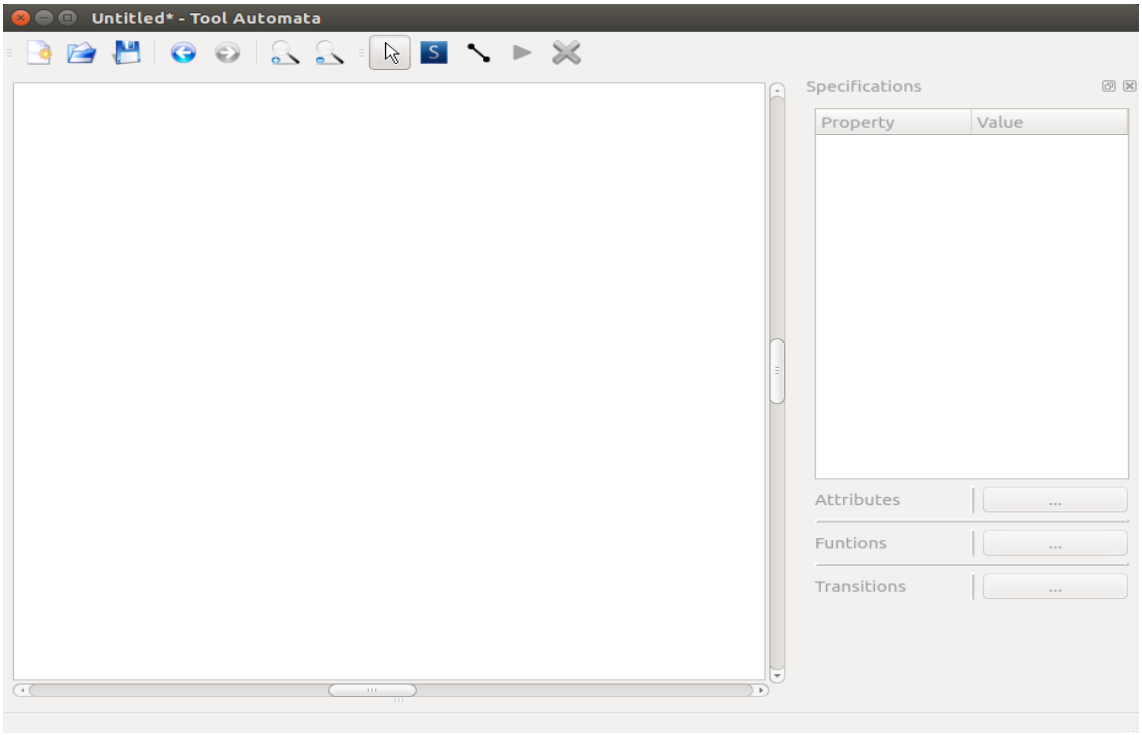


Figura 7 Interfaz principal

2.6.1 Principios para el diseño de interfaces

El diseño de interfaces de usuario es una tarea que ha adquirido relevancia en el desarrollo de un sistema. La calidad de la interfaz de usuario puede ser uno de los motivos que conduzca a un sistema al éxito o al fracaso. Para el diseño de la interfaz se utilizaron diferentes principios que se enuncian a continuación:

Anticipación: La aplicación debe intentar anticiparse a las necesidades del usuario y no esperar a que el usuario tenga que buscar la información, recopilarla o invocar las herramientas que va a utilizar. (25)

Autonomía: La computadora, la interfaz de usuario y el entorno de trabajo deben estar a disposición del usuario. Se debe dar al usuario un ambiente flexible para que pueda aprender rápidamente a usar la aplicación. (25)

Ley de Fitt: El tiempo para alcanzar un objetivo es una función de la distancia y tamaño del objetivo. Es por ello, que es conveniente usar objetos grandes para las funciones importantes. (25)

Objetos de Interfaz Humana: Los objetos de interfaz humana no son necesariamente los objetos que se encuentran en los sistemas orientados a objetos. Estos pueden ser vistos, escuchados, tocados o percibidos de alguna forma. Además, estos objetos deberían ser entendibles, consistentes y estables. (25)

2.6.2 Tratamiento de excepciones

El tratamiento de errores facilita el buen funcionamiento de la aplicación. Una excepción es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias por lo que cuando se produce un error debido a la entrada o sencillamente la manipulación incorrecta de algún dato por parte del usuario. Cuando esto sucede se muestra un mensaje de error que le permita al usuario percatarse de que la acción que realizó es incorrecta y pueda corregir su error. (26) En la herramienta se controlan los errores usando las técnicas de mensajes de alerta. A continuación se muestra el diseño de estos mensajes:

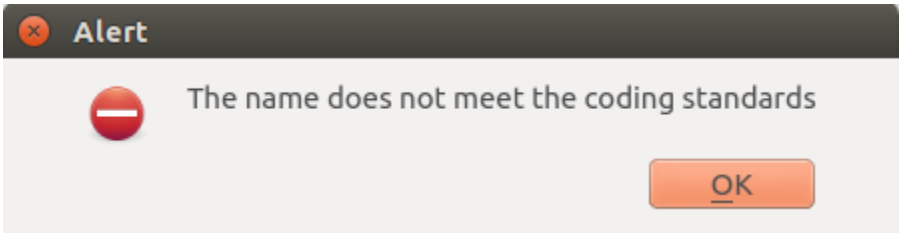


Figura 8 Tratamiento de excepciones

Para las acciones que son irreversible también se muestran mensajes de confirmación como el caso de cerrar la herramienta, por lo que se propuso la siguiente interfaz.

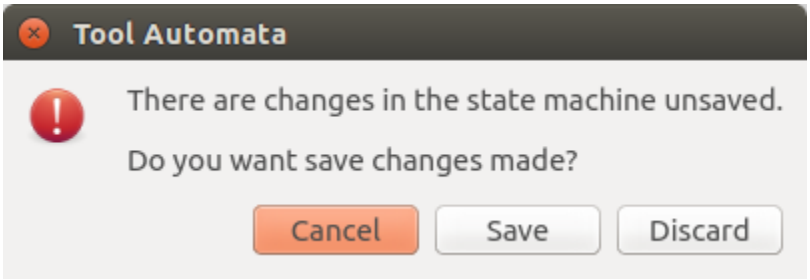


Figura 9 Utilización de mensajes de confirmación

2.7 Patrón arquitectónico

Un patrón arquitectónico define la estructura básica de una aplicación, provee un subconjunto de subsistemas predefinidos, incluyendo reglas, lineamientos para conectarlos, pautas para su organización y constituye una plantilla de construcción. (27)

Entre las ventajas del uso de patrones, se pueden encontrar:

- Permiten la reutilización de soluciones arquitectónicas de calidad.
- Son de gran ayuda para controlar la complejidad de un diseño.
- Facilitan la documentación de diseños arquitectónicos.
- Proporcionan un vocabulario común que mejora la comunicación entre diseñadores.

2.7.1 Patrón arquitectónico N-Capas

Se propone el uso del patrón arquitectónico N-Capas para el desarrollo de la aplicación, ya que permitirá hacer la implementación de forma más organizada. Aquí aparecen dos conceptos importantes: las capas que se encargan de la distribución lógica de los componentes, y los niveles que se refieren a la colocación física de los recursos. La característica principal de este patrón es que cada capa oculta las capas inferiores de las siguientes superiores a esta. (28)

Algunas de las ventajas de utilizar este patrón son (29):

- Mejoras las posibilidades de mantenimiento, debido a que cada capa es independiente de la otra, los cambios o actualizaciones pueden ser realizados sin afectar la aplicación como un todo.

- Flexibilidad, como cada capa puede ser manejada y escalada de forma independiente, la flexibilidad se incrementa.

- Disponibilidad, las aplicaciones pueden aprovechar la arquitectura modular de los sistemas habilitados usando componentes que escalan fácilmente lo que incrementa la disponibilidad.

La descomposición en capas que se propone es:

- **Capa de presentación:** Esta capa es la encargada de presentar el sistema al usuario, le comunica la información y captura la información en un mínimo de proceso. Esta capa se comunica únicamente con la capa de negocio. En esta se encuentran las clases encargadas de generar las interfaces gráficas y realizar validaciones a los datos entrados por el usuario.

- **Capa de negocio:** Aquí es donde se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados.

A continuación se muestra el diseño de la arquitectura de la aplicación utilizando el estilo 2 capas:

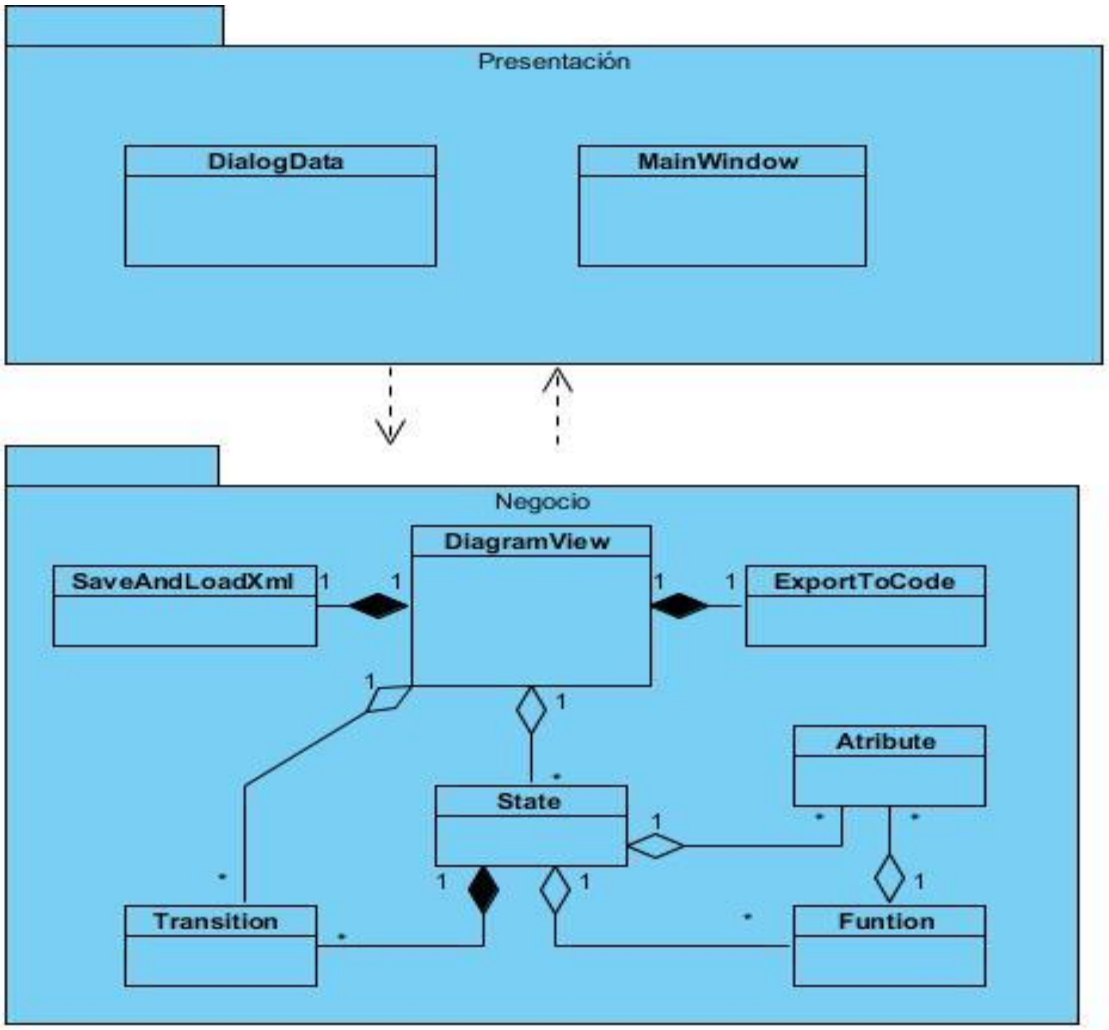


Figura 10 Diseño de la herramienta por capas

2.8 Patrones de diseño

La base de las soluciones a problemas comunes en el desarrollo de software son los patrones de diseño, su aplicación facilita el trabajo en el momento de implementar una aplicación. Los patrones que se emplean en la solución son parte del conjunto de patrones GRAPS, su utilización permitió refinar el

diseño y asignar las responsabilidades de las distintas clases, haciéndolas más sencillas, reutilizables y encapsuladas. A continuación se explican los patrones.

Alta Cohesión: Cada elemento del diseño debe realizar una labor única dentro del sistema, lo cual expresa que la información que almacena una clase debe de ser coherente y estar en la mayor medida de lo posible relacionada con la clase. En la perspectiva del diseño orientado a objetos, la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una clase con alta cohesión tiene responsabilidades estrechamente relacionadas y poco complejas. (30) Por ejemplo la clase *DiagramView* posee los métodos necesarios para satisfacer sus necesidades, sin sobrecargarla con métodos que no tengan que ver directamente con ella.

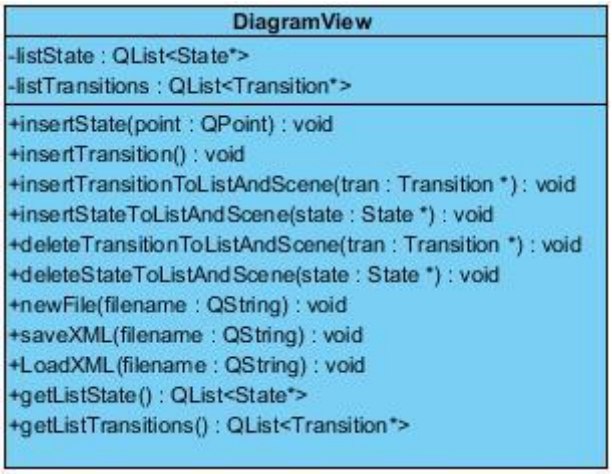


Figura 11 Ejemplo en la aplicación de patrón de alta cohesión

Creador: El patrón creador ayuda a identificar quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases. Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización. La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. (30) Por ejemplo la clase *DiagramView* necesita tener una instancia de la clase *ExportXML* para realizas sus funciones.

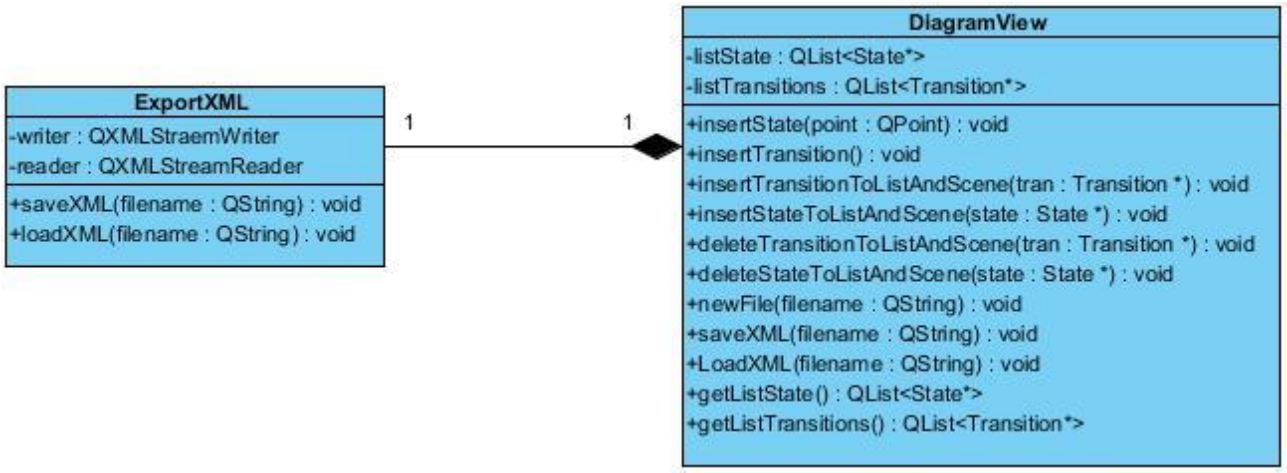


Figura 12 Ejemplo en la aplicación del patrón Creador

Controlador: Es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto es para aumentar la reutilización de código y a la vez tener un mayor control. Recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento. (30) Por ejemplo la clase *DiagramView* es la encargada de manejar los eventos de la clase *ExportXML*.

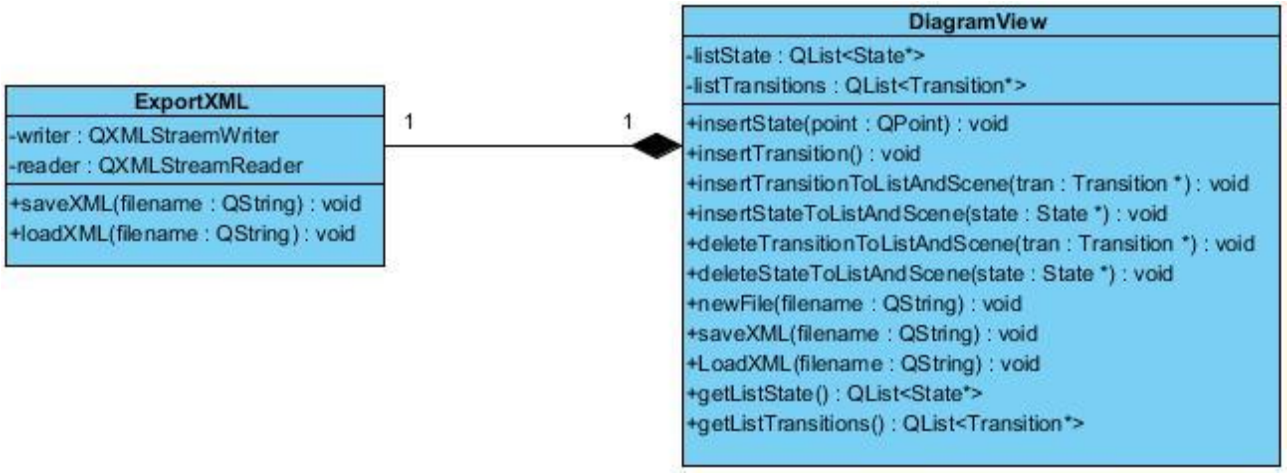


Figura 13 Ejemplo en la aplicación del patrón Controlador

Bajo acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está relacionada a otras clases. Una clase con bajo o débil acoplamiento no depende de muchas otras. Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una

modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. La solución es asignar las responsabilidades de forma tal que las clases se comuniquen con el menor número de clases que sea posible, sin comprometer la funcionalidad por supuesto. (30)Por ejemplo la clase *ExportXML* solo depende de la clase *DiagramView* para realizar sus funciones.

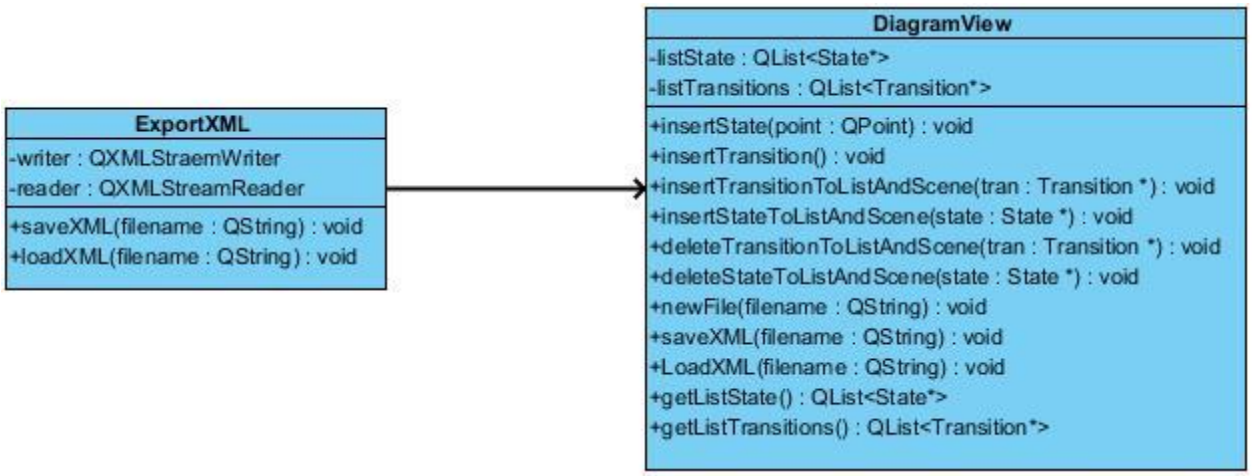


Figura 14 Ejemplo en la aplicación del patrón de Bajo Acoplamiento

Experto: Un experto es una clase que tiene toda la información necesaria para implementar una responsabilidad. La respuesta es asignar la responsabilidad a la clase que contenga la información necesaria para cumplir la responsabilidad, o sea, la clase debe ser la experta en la información. La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). (30) Por ejemplo la clase *DiagramView* tiene la información necesaria para obtener los datos de la clase *State*.

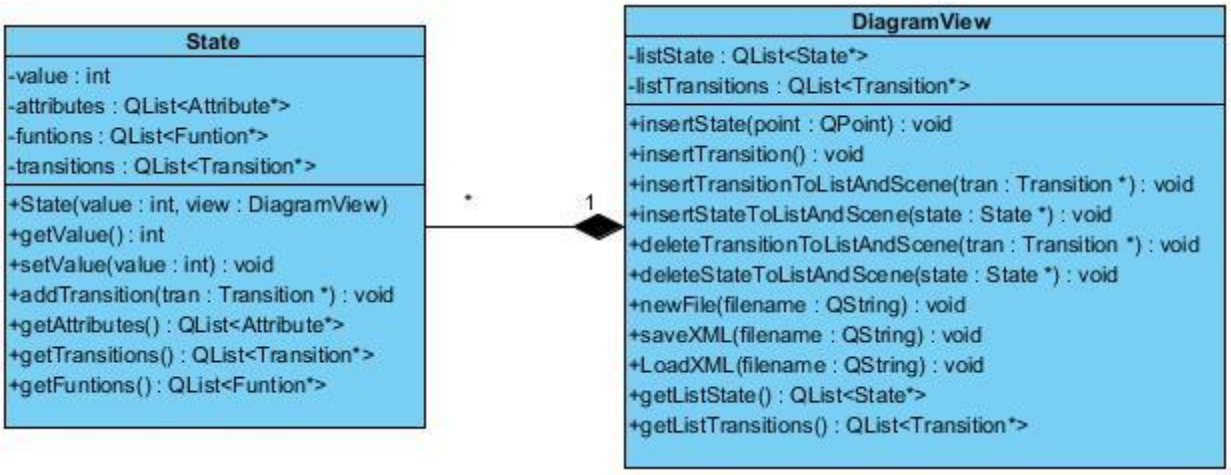


Figura 15 Ejemplo en la aplicación del patrón Experto

2.9 Diseño de la solución propuesta

La Metodología XP plantea que la implementación de un software debe realizarse de forma iterativa, obteniendo al culminar cada iteración un producto funcional que debe ser probado y mostrado al cliente para incrementar la visión de los desarrolladores con la opinión de éste. Esta es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.

En el diseño de las aplicaciones realizadas bajo las reglas de la metodología XP, no es requerida la representación del sistema mediante diagramas de clase utilizando notación UML, sino que se utilizan otras técnicas, como las llamadas tarjetas CRC (Contenido, Responsabilidad y Colaboración). Sin embargo la utilización de estos diagramas puede aplicarse siempre y cuando influya en el mejoramiento de la comunicación entre el equipo de desarrollo y se enfoquen en la información importante. (24)

2.9.1 Tarjetas CRC

Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Además permiten que el equipo completo contribuya en la tarea del diseño. (24)

Tabla 20 Tarjeta CRC Attribute

Tarjeta CRC	
Class Attribute	
Responsabilidades	**Colaboradores**

Esta clase es la encargada de almacenar los datos de un atributo.
 	
public:
Atribute(QString pname="newAttribute",QString ptype="", QString pvisibility="private", QString pinicial_value="");
QString getName();
QString getType();
QString getVisibility();
QString getInicialValue();
void setName(QString nam);
void setType(QString typ);
void setVisibility(QString visib);
void setInicialValue(QString inicial_v);
 	
private:
QString name,
QString type, visibility,
QString inicial_value;
	Class dialog-datos.h
	Class funtion.h
	Class state.h
	Class table-model.h

QString value;	

Tabla 21 Tarjeta CRC SaveAndLoad

Tarjeta CRC	
Class SaveAndLoadXML	
Responsabilidades	**Colaboradores**

Esta clase es la encargada de importar y exportar los archivos en formato XML.
 	
public:
SaveAndLoadXML(DiagramView * view);
saveXML(QString filename);
loadXML(QString filename);
 	
private:
DiagramView * view
QXMLStreamWriter * writer;
QXMLStreamReader * reader;
	Class diagram-view.h
	Class state.h

Tabla 22 Tarjeta CRC DiagramView

Tarjeta CRC	
class DiagramView	
Responsabilidades	**Colaboradores**

Esta clase es la encargada de gestionar los estados y gestionar transiciones de la máquina de estados.
 	
public:
DiagramView(MainWindow *parent = 0);
void insertState(QPoint pos);
void insertTransition();
void insertTransitionToListAndScene(Transition* tra);
void insertStateToListAndScene(State* state);
void deleteTransitionToListAndScene(Transition* tra);
void deleteStateToListAndScene(State* tra);
void newFile();
bool isSaved(){return saved;}
bool isChanged() const;
void setInitial(State*state);
void setEnd(State* state);
QList<State*> getListStates(){return listStates;}
QList<Transition*> getListTransitions(){return listTransitions;}
void showTabWidget(int tab);
 	
public slots:
void canUndoChangedSlot(bool can);
	Class commands.h
	Class main-window.h
	Class state.h
	Class transition.h
	Class save-and-load.h
	Class export-to-XML.h

void canRedoChangedSlot(bool can);
void deleteItem();
void setClean(bool clean);
void exportCodeQt(QString dir);
 	
private:
Mode myMode;
MouseAction myMouseAction;
QList<State*> listStates;
QList<Transition*> listTransitions;
int ctdStates;
QString fileName;
QUndoStack *undoStack;	

Tabla 23 Tarjeta CRC Funtion

Tarjeta CRC	
class Funtion	
Responsabilidades	**Colaboradores**

Esta clase es la encargada de almacenar los datos de una función.
 	
public:
Funtion(QList<Attribute*> plistAttribute,QString name= "newFuntion", QString ptype="", QString pvisibility="public")
QString getName;
QString getType;
QString getVisibility;
QList<Attribute*> getListAttribute;
void setName(QString nam;
void setType(QString type);
void setVisibility(QString visib);
void setListAttribute(QList<Attribute*>list);
 	
private:
QString name;
QString type;
QString visibility;
QList<Atribute*> listAttribute;
	Class table-model.h
	Class state-h
	Class Attribute-h

Tabla 24 Tarjeta CRC ExportXML

Tarjeta CRC	
Class ExportToCode	
Responsabilidades	**Colaboradores**
Esta clase es la encargada de exportar la máquina de estados a código fuente C++. public: ExportToCode(DiagramView * view);	Class diagram-view.h Class state.h

exportQtCode(QString dir); private: DiagramView * view; QTextStream writer;	

Tabla 25 Tarjeta CRC Transition

Tarjeta CRC	
class Transition	
Responsabilidades	**Colaboradores**

Esta clase es la encargada de guardar los datos de una transición.
 	
public:
Transition(State *ss, State *es, QGraphicsScene *scene=0);
State* getStartState();
State* getEndState();
bool hasEndState();
void setStartState(State *state);
void setEndState(State *state);
Text * getText();
State *startState, *endState;
Text * text;
	Class state.h

Tabla 26 Tarjeta CRC MainWindow

Tarjeta CRC	
class MainWindow	
Responsabilidades	**Colaboradores**

Esta clase es donde se encuentra implementada la interfaz principal
 	
private slots:
void makePhoto();
void itemClicked(QGraphicsItem *item);
void valueChanged(QtProperty *property, const QVariant &value);
void newFile();
bool save();
bool saveAs();
void openFile();
void exportCode();
void startAutomaticTest();
void startManualTest(State *state);
 	
private:
class QtVariantPropertyManager *variantManager;
class QtTreePropertyBrowser *propertyEditor;
void addProperty(QtProperty *property, const QString &id);
	Class diagram-view.h

QMap<QtProperty *, QString> propertyToId; QMap<QString, bool> idToExpanded; QMap<QString, QtProperty *> idToProperty; DiagramView * diagramView;	

Tabla 27 Tarjeta CRC State

Tarjeta CRC	
class State	
Responsabilidades	**Colaboradores**

Esta clase es la encargada de almacenar los datos de un estado
 	
public:
State(int position, DiagramView *pview = 0);
void setValue(int pvalue);
int getValue();
void setChecked(bool checked);
void removeTransition(Transition *transition);
void removeTransitions();
void addTransition(Transition *transition);
void setListAtri(QList<Attribute*> list);
void setListFunt(QList<Funtion*> list);
void setListTran(QList<Transition*> list);
QList<Transition*>getTransitions();
QList<Funtion*> getFuntions();
QList<Attribute*> getAttributes ();
Text * getText();
 	
private:
DiagramView * view;
QPointF oldPos;
bool checked;
QBrush color;
QList<Transition*> transitions;
QList<Attribute*> attributes;
QList<Funtion*> funtions;
Text * text;
int value;
	Class commands.h
	Class diagram-view.h
	Class transition.h

Tabla 28 Tarjeta CRC TableView

Tarjeta CRC	
class TableView	
Responsabilidades	**Colaboradores**
Esta clase es la encargada de crear una tabla que permite gestionar los atributos, funciones y transiciones de los estados	Class transition.h Class state.h

public:
TableView(TableType type,QList<Attribute*> list, QWidget *parent=0);
TableView(QList<Funtion *> list, QWidget *parent=0);
TableView(QList<Transition*> list, DiagramView* view, QWidget *parent=0);
ModelTable *getModel(){return model;}
 	
signals:
void buttonEnable();
void buttonNoEnable();
 	
public slots:
void deleteRow();
 	
private:
QWidget * widgetParent;
TableType tableType;
ModelTable * model;	

Tabla 29 Tarjeta CRC Text

Tarjeta CRC	
class Text	
Responsabilidades	**Colaboradores**
Esta clase es la encargada de almacenar texto y la posición del texto con respecto al padre (estado o transición). public: Text(QGraphicsItem *parent, DiagramView *view = 0); QPointF posParam() const { return posText; } void setPosParam(QPointF posParam) { posText = posParam; } private: QPointF posText; QString oldText; QString newText;	Class transition.h Class state.h

2.10 Estructura del Fichero XML

El fichero de formato XML mediante el cual se exportan e importan los datos de valor para la herramienta en cuestión tiene una estructura diseñada para esta herramienta (Ver en los anexos), a continuación se enuncian las etiquetas y los datos que contienen:

<STATE>: contiene las etiquetas <VALUE>, <X>, <Y>, <TEXT>, <BRUSH>, <FONT>, <PEN>, <ATTRIBUTES> y <FUNTIONS>, estas etiquetas almacenan los datos de un estado.

<TRANSITION>: contiene las etiquetas <NAME>, <START>, <END>, <FONT> Y <PEN>, estas etiquetas almacenan los datos de una transición.

<VALUE>: es un número único que identifica a cada estado.

<X>: es un número que almacena la posición del estado en el eje de coordenadas X del área de trabajo.

<Y>: es un número que almacena la posición del estado en el eje de coordenadas Y del área de trabajo.

<TEXT>: es una cadena de caracteres que indica el nombre de un estado.

<INITIAL>: es una cadena de caracteres que puede ser “true” o “false”, indica si un estado es inicial.

<START>: es un número que almacena el valor identificativo del estado de donde inicia la transición.

<END>: dentro de la etiqueta <STATE> es una cadena de caracteres que puede ser “true” o “false”, indica si un estado es final, dentro de la etiqueta <TRANSITION> almacena el número identificativo del estado donde termina la transición.

<BRUSH>: es una cadena de caracteres en el formato #XXXXXX, este dato almacena el color de relleno de un estado.

<FONT>: contiene las etiquetas <TYPE>, <SIZE>, <BOLD>, <ITALIC>, <UNDERLINE> y <KERNINIG>, estas etiquetas almacenan los datos del formato del texto de un estado o una transición.

<TYPE>: es una cadena de caracteres que almacena el tipo de letra.

<SIZE>: es un número que almacena el tamaño de la letra.

<BOLD> es una cadena de caracteres que puede ser “true” o “false”, indica si el formato de la letra es negrita.

<ITALIC> es una cadena de caracteres que puede ser “true” o “false”, indica si es *cursiva.*

<UNDERLINE> es una cadena de caracteres que puede ser “true” o “false”, indica si es subrayada.

<STRIKEOUT> es una cadena de caracteres que puede ser “true” o “false”, indica si es tachado.

<KERNING> es una cadena de caracteres que puede ser “true” o “false”, indica si la separación entre caracteres es mucha o poca.

<PEN>: contiene las etiquetas <COLOR>, <WIDTH> y <STYLE>, estas etiquetas almacenan los datos para dibujar un estado o una transición en el área de trabajo.

<COLOR>: es una cadena de caracteres en el formato #XXXXXX, este dato almacena el color de la línea de un estado o una transición.

<WIDTH>: es un número que almacena el ancho de la línea de un estado o una transición.

<STYLE>: es un número que almacena el tipo de estilo de la línea de un estado o una transición.

<ATTRIBUTE>: contiene las etiquetas <NAME>, <TYPE>, <VISIBILITY> y <INITIAL_VALUE>, estas etiquetas almacenas los datos de un atributo.

<NAME>: contiene una cadena de caracteres que almacena el nombre de un atributo, parámetro, función o transición.

<TYPE>: contiene una cadena de caracteres que almacena el tipo de dato de un atributo o parámetro o el tipo de dato que retorna una función.

<VISIBILITY> contiene una cadena de caracteres que almacena la visibilidad de un atributo o una función.

<INITIAL_VALUE> contiene una cadena de caracteres que almacena el valor inicial de un atributo o parámetro.

<FUNTION>: contiene las etiquetas <NAME>, <TYPE>, <VISIBILITY> y <PARAMETERS>, estas etiquetas almacenan los datos de una función.

<PARAMETER>: contiene las etiquetas <NAME>, <TYPE> y <INITAIL_VALUE>, estas etiquetas almacenan los datos de un parámetro.

<PARAMETERS>: almacena la etiqueta <PARAMETER> que aparecerá según la cantidad de parámetros que tenga la función, por ejemplo:

```
<PARAMETERS>
    <PARAMETER>
        :
    </PARAMETER>
    <PARAMETER>
        :
    <PARAMETER>
    :
</PARAMETERS>
```

<TRANSITONS>: tiene la misma estructura que la etiqueta <PARAMETERS>, en vez de <PARAMETER> es la etiqueta <TRANSITION>.

<STATES>: tiene la misma estructura que la etiqueta <PARAMETERS>, en vez de contener la etiqueta <PARAMETER> es la etiqueta <STATE>.

<ATTRIBUTES>: tiene la misma estructura que la etiqueta <PARAMETERS>, en vez de contener la etiqueta <PARAMETER> es la etiqueta <ATTRIBUTE>.

<FUNTIONS>: tiene la misma estructura que la etiqueta <PARAMETERS>, en vez de contener la etiqueta <PARAMETER> es la etiqueta <FUNTION>.

2.11 Estructura de la máquina de estados que se exporta

La funcionalidad de exportar a código C++ la máquina de estados diseñada exporta el código con la estructura que se muestra a continuación:

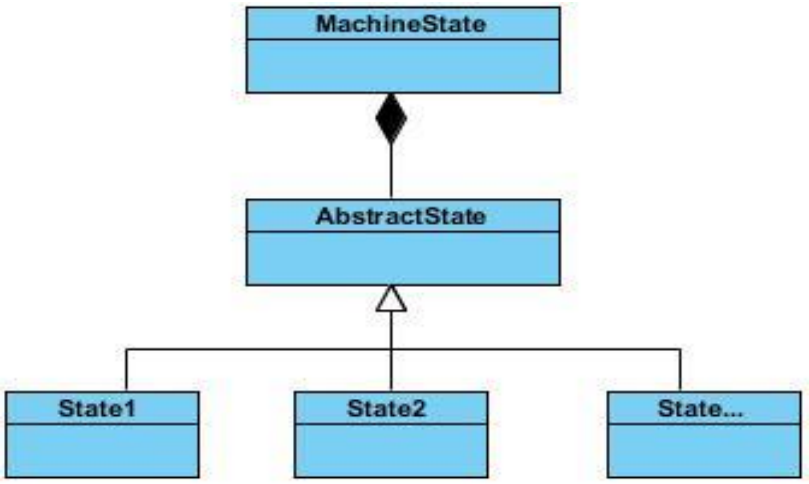


Figura 16 Estructura de la máquina de estados exportada a código C++

Consideraciones Parciales

En este capítulo se escogieron las tecnologías para la implementación de la herramienta, C++ como lenguaje de programación, XP como metodología de desarrollo y XML para guardar y cargar la configuración de las máquinas de estado. También se definieron los requisitos necesarios para la implementación. Se describieron además los requisitos mediante las HU. Se realizó la estimación del tiempo por semanas con un total de 15, donde se precisó también el plan de iteraciones que define en qué momento se implementa cada requisito. Finalmente se describió el patrón arquitectónico N-capas que fue aplicado a la solución y se especificaron las tarjetas CRC para lograr mejor entendimiento de la herramienta.

CAPÍTULO 3 IMPLEMENTACIÓN Y PRUEBA

3. Introducción

En el presente capítulo se establecen criterios asociados a la implementación y validación del sistema propuesto. Se somete la herramienta a una serie de pruebas de aceptación para detectar defectos en el software, verificar la integración adecuada de los componentes así como verificar que todos los requisitos se han implementado correctamente garantizando su correcto funcionamiento.

3.1 Estándares de codificación

Un estándar de programación es una forma de "normalizar" la programación de forma tal que al trabajar en un proyecto cualquiera de las personas involucradas en el mismo tenga acceso y comprenda el código. En otras palabras define la escritura y organización del código fuente de un programa. Además el seguir un estándar de programación te facilita como programador la modificación de tu propio código fuente aunque no estés trabajando en un equipo. Por lo general los estándares de programación definen la forma en que deben ser declaradas las variables, las clases, los comentarios, en algunos estándares se especifica que datos deben incluirse acerca del programador y de los cambios realizados al código fuente, etc.

Para el desarrollo de la solución informática se utilizó “Estándares de codificación para el lenguaje C++ en el Centro De Diseño y Simulación de Estructuras Mecánicas”, con el objetivo de lograr uniformidad en el código y facilitar la comprensión.

3.1.1 Declaración de los espacios de nombre y las clases

Convenio de nombres

Se utilizará el idioma inglés para denotar el nombre de los archivos. De igual forma se utilizará este idioma para denotar todos los espacios de nombres, clases, métodos, variables constantes, y de manera general. En caso de no conocer la palabra correcta en idioma inglés se podrá utilizar, una palabra en español.

Espacio de nombres y de clases

Los identificadores para los espacios de nombres y las clases siempre serán sustantivos y comenzarán además con letra inicial mayúscula. El resto del nombre será en minúscula.

```
class TableView:public QTableView
{

};
```

Atributos de clases, parámetros, variables locales

Todos los caracteres con se escribirán en minúscula. En caso de que existan nombre compuestos se utiliza la siguiente regla. El primer nombre se escribe con minúscula y a partir de ese momento cada nuevo nombre con letra inicial mayúscula.

```
class TableView:public QTableView
{

   private:
    QWidget * widgetParent;
    TableType tableType;
    ModelTable * model;
    QPushButton *parameter;
};
```

Directivas de pre-procesamiento

Son la única excepción para la cual se utilizarán *underscores* y mayúsculas. Se tratará siempre de prefijar los identificadores de las directivas para agruparlas lógicamente por función, área u otra dimensión.

```
#define TABLEVIEW_H
```

Clases, espacios de nombre, implementación de métodos

table-view.h

```
class TableView:public QTableView
{
  Q_OBJECT
  public:
    enum TableType {ATRIBUTE, FUNTION, TRANSITION, PARAMETER};
    TableView (TableType type, QList<Atribute*> list, QWidget *parent=0);
    TableView (TableType type, QList<Funtion*> list, QWidget *parent=0);
    TableView (TableType type, QList<Transition*> list, QWidget *parent=0);
    ModelTable *getModel () {return model ;}

  protected:
    void mousePressEvent (QMouseEvent * event);
    void resizeEvent (QResizeEvent *event);

  signals:
    void buttonEnable ();
    void buttonNoEnable ();
```

```
  public slots:
    void deleteRow ();

  private:
    QWidget * widgetParent;
    TableType tableType;
    ModelTable * model;
    QPushButton *parameter;
};
```

table-view.cpp

```
TableView::TableView(TableType type ,QList<Atribute *> list, QWidget *parent)
    :QTableView(parent)
{
    if(type==ATRIBUTE)
      {
         model= new ModelTable(list,ModelTable::ATRIBUTE, this);
      }
      else
        {
           model= new ModelTable(list,ModelTable::PARAMETER, this);
        }
    this->setModel(model);
    this->setSelectionBehavior(QAbstractItemView::SelectRows);
    this->setSelectionMode(QAbstractItemView::SingleSelection);
}
```

3.2 Implementación

En la fase de implementación, XP plantea la implementación de cada una de las HU. Al inicio se realiza un chequeo de cada una de las HU junto con el plan de iteraciones y se modifica en caso de ser necesario. Se crean tareas para ayudar a organizar la implementación de las HU asignando a un desarrollador la responsabilidad de su implementación. Estas tareas normalmente se escriben en un lenguaje técnico a diferencia de las HU que son escritas en el lenguaje del cliente. La fase de implementación incluye 4 iteraciones de desarrollo sobre el sistema, obteniéndose el producto deseado. A continuación se detalla cada una de las iteraciones.

Iteración 1

En esta iteración se comenzó con la configuración de la herramienta y se inicia la implementación de las funcionalidades que permiten gestionar estados y conexiones, además de seleccionar y mover los elementos en la escena.

Tabla 30 Historias de usuario abordadas en la primera iteración

Historias de usuario	Estimación (semanas)	Real (semanas)
Insertar estados	1/4	1/4
Modificar estados	2/4	2/4
Eliminar estados	1/4	1/4
Seleccionar uno o varios elementos en la escena	1	1
Insertar estados	1/4	1/4
Modificar estados	2/4	2/4
Eliminar estados	1/4	1/4
Mover los elementos en la escena	1	1
Deshacer y rehacer	1	1

Tabla 31 Tarea para la HU # 1

Tarea	
No de tarea: 1	**No de HU:** HU1
Nombre de la tarea: implementación de insertar, modificar y eliminar estados	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 27/1/2014	**Fecha fin:** 2/2/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se crearán las funcionalidades que permitirán al desarrollador insertar, modificar y eliminar estados. De esta manera la herramienta posibilita la agregación, modificación y eliminación de cada estado en la escena	

Tabla 32 Tarea para la HU # 2

Tarea	
No de tarea: 1	**No de HU:** HU2
Nombre de la tarea: implementación de seleccionar	
Tipo de tarea: Desarrollo	**Estimación:** 1

Fecha de inicio: 3/2/2014	**Fecha fin:** 9/2/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se creará la funcionalidad que permitirá al desarrollador seleccionar uno o varios elementos de la escena	

Tabla 33 Tarea para la HU # 3

Tarea	
No de tarea: 1	**No de HU:** HU3
Nombre de la tarea: implementación de insertar, modificar y eliminar conexiones	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 10/1/2014	**Fecha fin:** 16/2/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se crearán las funcionalidades que permitirán al desarrollador insertar, modificar y eliminar conexiones entre los estados de la escena. De esta manera la herramienta posibilita la agregación, modificación y eliminación de cada conexión en la escena	

Tabla 34 Tarea para la HU # 4

Tarea	
No de tarea: 1	**No de HU:** HU4
Nombre de la tarea: implementar la funcionalidad de mover los elementos de la escena	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 17/1/2014	**Fecha fin:** 23/2/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se creará la funcionalidad que permitirá al desarrollador mover uno o varios estados y actualizar automáticamente las conexiones que entra y salen de los mismos	

Tabla 35 Tarea para la HU # 5

Tarea	
No de tarea: 1	**No de HU:** HU5
Nombre de la tarea: implementar la funcionalidad de deshacer y rehacer	

Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 24/2/2014	**Fecha fin:** 2/3/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se crearán las funcionalidades que permitirán al desarrollador deshacer y rehacer las acciones que se visualicen en la escena	

Iteración 2

En esta iteración se continuó con la implementación de las funcionalidades que permiten gestionar las especificaciones de los estados, realizar el zoom en la escena y guardar una imagen del diseño, después de esta iteración se tiene un producto funcional.

Tabla 36 Historias de usuario abordadas en la segunda iteración

Historia de usuario	Estimación (semanas)	Real (semanas)
Gestionar características de los estados	2	2
Permitir el zoom en la escena	1	1
Guardar imagen del diseño	1	1

Tabla 37 Tarea para la HU # 6

Tarea	
No de tarea: 1	**No de HU:** HU6
Nombre de la tarea: implementar las funcionalidades de gestionar las especificaciones de los estados	
Tipo de tarea: Desarrollo	**Estimación:** 2
Fecha de inicio: 3/3/2014	**Fecha fin:** 14/3/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se crearán las funcionalidades que permitirán al desarrollador agregar, modificar y eliminar atributos, funciones con sus parámetros y modificar y eliminar las conexiones de cada estado	

Tabla 38 Tarea para la HU # 7

Tarea	
No de tarea: 1	**No de HU:** HU7

Nombre de la tarea: implementar las funcionalidades que permitan realizar el zoom en la escena	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 17/3/2014	**Fecha fin:** 23/3/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se crearán las funcionalidades que permitirán al desarrollador alejar o acercar la escena para tener una mejor vista del diseño	

Tabla 39 Tarea para la HU # 8

Tarea	
No de tarea: 1	**No de HU:** HU8
Nombre de la tarea: implementar de la funcionalidad guardar imagen del diseño	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 24/3/2014	**Fecha fin:** 30/3/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se crearán las funcionalidades que permitirán al desarrollador guardar una imagen del diseño de la máquina de estados	

Iteración 3

En esta iteración se continuó con la implementación de las funcionalidades que permiten guardar, cargar y exportar.

Tabla 40 Historias de usuario abordadas en la tercera iteración

Historia de usuario	Estimación (semanas)	Real (semanas)
Guardar el diseño de la máquina de estados en un archivo en formato XML	2	2
Cargar el diseño de la máquina de estados desde un archivo en formato XML	1	1/2
Exportar a código C++	2	2.5

Tabla 41 Tarea para la HU # 9

Tarea	
No de tarea: 1	**No de HU:** HU9
Nombre de la tarea: implementar de la funcionalidad guardar el diseño	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 31/3/2014	**Fecha fin:** 12/4/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se creará la funcionalidad que permitirá al desarrollador guardar el diseño en formato XML para trabajar luego en el mismo diseño, el archivo se guardará en la dirección deseada por el usuario.	

Tabla 42 Tarea para la HU # 10

Tarea	
No de tarea: 1	**No de HU:** HU10
Nombre de la tarea: implementar de la funcionalidad cargar un diseño	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 21/4/2014	**Fecha fin:** 24/4/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se creará la funcionalidad que permitirá al desarrollador cargar un autómata previamente creado y guardado, permitirá cargar el archivo desde cualquier dirección	

Tabla 43 Tarea para la HU # 11

Tarea	
No de tarea: 1	**No de HU:** HU11
Nombre de la tarea: implementar de la funcionalidad exportar un diseño a código C++	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 25/4/2014	**Fecha fin:** 11/5/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se creará la funcionalidad que permitirá al desarrollador exportar el diseño de la máquina de estados a código fuente C++, con la estructura que señala el patrón State, permitiendo al usuario exportarlo donde desee.	

Iteración 4

En la cuarta y última iteración se desarrolló la funcionalidad de probar el funcionamiento del autómata. Al finalizar se cuenta con un producto listo para poner en funcionamiento.

Tabla 44 Historias de usuario abordadas en la cuarta iteración

Historia de Usuario	Estimación (semanas)	Real (semanas)
Probar el funcionamiento de un autómata	3	3

Tabla 45 Tarea para la HU # 12

Tarea	
No de tarea: 1	**No de HU:** HU12
Nombre de la tarea: implementar de la funcionalidad probar el funcionamiento de un autómata	
Tipo de tarea: Desarrollo	**Estimación:** 1
Fecha de inicio: 12/4/2014	**Fecha fin:** /5/2014
Programador responsable: Luis Kayrumet Pérez Buigas	
Descripción: se creará la funcionalidad que permitirá al desarrollador probar el funcionamiento de un autómata, donde podrá probar las transiciones, así como observar si el autómata sigue los estados que desea el usuario.	

3.3 Pruebas de Software

Unas de las vías más importantes para determinar el estado de la calidad de un producto de software es el proceso de pruebas. Estas están dirigidas a componentes del sistema en su totalidad, con el objetivo de medir el grado en que cumple con los requerimientos. En ellas se usan casos de prueba, especificados de forma estructurada mediante técnicas. Sus objetivos, métodos y técnicas usadas se describen en el plan de prueba. (31)

La prueba es una actividad fundamental en muchos procesos de desarrollo, incluyendo el del software. Estas permiten detectar la presencia de errores que pudieran generar las entradas o salidas de datos y comportamientos inapropiados durante su ejecución. (31)

Al aplicarles las pruebas al software se deben seguir un conjunto de estrategias para lograr que estas se hagan en el menor tiempo posible y con la calidad requerida, además de garantizar que arrojen los resultados esperados. (31)

La metodología XP plantea dos grupos de pruebas: las unitarias, que las realizan los programadores, estas pruebas son las encargadas de verificar el código de forma automática. El otro grupo de pruebas son las de aceptación, que están dedicadas a probar si al final de una iteración se obtuvieron las funcionalidades especificadas y además que den las respuesta como el usuario desea.

3.3.1 Pruebas de Aceptación

El uso de cualquier producto de software tiene que estar justificado por las ventajas que ofrece. Sin embargo, antes de su puesta en marcha es muy difícil determinar si sus ventajas realmente justifican su uso. El mejor instrumento para esta determinación es la llamada "prueba de aceptación". En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique y eliminar la influencia de conflictos de intereses, y para que sea lo más objetiva posible, la prueba de aceptación no debería ser responsabilidad de los ingenieros de software que han desarrollado el producto. (31)

Estas pruebas las realiza el cliente. Son básicamente pruebas funcionales sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual del usuario. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado e integrado o pudiera ser una versión del producto o una iteración funcional pactada previamente con el cliente. (31)

Tabla 46 Prueba de aceptación para la HU “Insertar estados”

Caso de Prueba de Aceptación	
Código: HU_1_P1	**Historia de Usuario:** 1
Nombre: Insertar estados	
Descripción: Prueba para la funcionalidad de Insertar estado	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y debe dar la opción de insertar estados	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de insertar estado • Se da clic en el área de trabajo	
Resultado Esperado: Que la herramienta permita insertar estados, que permita modificar y eliminar estados existentes. La herramienta debe mostrar en la escena los estados insertados	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 47 Prueba de aceptación para la HU “Seleccionar uno o varios elementos en la escena”

Caso de Prueba de Aceptación

Código: HU_2_P1	**Historia de Usuario:** 2
Nombre: Seleccionar uno o varios elementos en la escena	
Descripción: Prueba para la funcionalidad de seleccionar estados	
Condiciones de Ejecución: La aplicación debe de estar en ejecución, debe dar la opción de seleccionar y deben existir elementos en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de seleccionar • Se da clic en el área de trabajo sobre un elemento	
Resultado Esperado: Que la herramienta permita seleccionar un elemento o varios elementos de la escena, y mostrar alrededor de los elementos seleccionados una línea discontinua	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 48 Prueba de aceptación para la HU “Insertar conexiones”

Caso de Prueba de Aceptación	
Código: HU_3_P1	**Historia de Usuario:** 3
Nombre: Insertar conexiones entre los estados	
Descripción: Prueba para la funcionalidad de insertar conexiones	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y deben existir uno o varios estados en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de insertar conexión • Se selecciona en el área de trabajo un estado y luego se selecciona el mismo estado u otro estado	
Resultado Esperado: Que la herramienta inserte una conexión de tipo ciclo sobre el estado seleccionado o una conexión de tipo arco en los dos estados seleccionados, además permita modificar y eliminar la conexión insertada.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 49 Prueba de aceptación para la HU “Mover los elementos en la escena”

Caso de Prueba de Aceptación	
Código: HU_4_P1	**Historia de Usuario:** 4
Nombre: Mover estados en la escena	
Descripción: Prueba para la funcionalidad de mover elementos	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y deben existir uno o varios estados en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de seleccionar • Se da clic en el área de trabajo sobre uno o sobre varios elementos presionando la tecla “Shift” para seleccionarlos. • Se da clic sobre el o uno de los objetos seleccionados y se arrastran en la escena	

Resultado Esperado: Que la herramienta permita mover los estados seleccionados dentro de la escena a donde es usuario desee
Evaluación de la Prueba: Prueba satisfactoria

Tabla 50 Prueba de aceptación para la HU "Deshacer y rehacer"

Caso de Prueba de Aceptación	
Código: HU_5_P1	**Historia de Usuario:** 5
Nombre: Deshacer y rehacer cambios en la escena	
Descripción: Prueba para la funcionalidad de deshacer cambios en la escena	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y deben haberse realizado uno o varios cambios realizados en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se realiza algún cambio en la escena • Se pulsa en el botón deshacer	
Resultado Esperado: Que la herramienta permita deshacer los cambios hechos en la escena	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 51 Prueba de aceptación para la HU "Gestionar características de los estados"

Caso de Prueba de Aceptación	
Código: HU_6_P1	**Historia de Usuario:** 6
Nombre: Insertar, modificar y eliminar atributos y funciones	
Descripción: Prueba para la funcionalidad de insertar atributos	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y deben existir uno o varios estados en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de seleccionar • Se da doble clic en el área de trabajo sobre un estado • La aplicación muestra una ventana que muestra una tabla • Se selecciona la opción de Agregar	
Resultado Esperado: Que la herramienta muestre la ventana donde está la tabla con los atributos del estado y permita agregar atributos a la tabla, la herramienta debe dar la posibilidad de modificarlos y eliminarlos	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 52 Prueba de aceptación para la HU "Gestionar características de los estados"

Caso de Prueba de Aceptación	
Código: HU_6_P2	**Historia de Usuario:** 6
Nombre: Insertar, modificar y eliminar atributos y funciones	
Descripción: Prueba para la funcionalidad de insertar funciones	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y deben existir uno o varios estados en la escena	

Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de seleccionar • Se da doble clic en el área de trabajo sobre un estado • La aplicación muestra una ventana que muestra una tabla • Se selecciona la opción de Agregar
Resultado Esperado: Que la herramienta muestre una ventana donde está la tabla con las funciones del estado y permita agregar funciones a la tabla, la herramienta debe dar la posibilidad de modificarlas y eliminarlas
Evaluación de la Prueba: Prueba satisfactoria

Tabla 53 Prueba de aceptación para la HU "Gestionar características de los estados"

Caso de Prueba de Aceptación	
Código: HU_6_P3	**Historia de Usuario:** 6
Nombre: Insertar, modificar y eliminar atributos y funciones	
Descripción: Prueba para la funcionalidad de insertar parámetros	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y deben existir uno o varios estados en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de seleccionar • Se da doble clic en el área de trabajo sobre un estado • La aplicación muestra una ventana que muestra una tabla • Se selecciona la opción de Agregar • La aplicación agrega una nueva función a la tabla • Se da clic en el botón parámetros de la función.	
Resultado Esperado: Que la herramienta muestra la ventana con una tabla donde están los parámetros de la función y permita agregar parámetros a la tabla, la herramienta debe dar posibilidad de modificarlos y eliminarlos	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 54 Prueba de aceptación para la HU "Permitir el zoom en la escena"

Caso de Prueba de Aceptación	
Código: HU_7_P1	**Historia de Usuario:** 7
Nombre: Realizar el zoom en la escena	
Descripción: Prueba para la funcionalidad de alejar la escena	
Condiciones de Ejecución: La aplicación debe de estar en ejecución	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de alejar	
Resultado Esperado: Que la herramienta aleje la escena y permita acercarla	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 55 Prueba de aceptación para la HU "Guardar imagen del diseño"

Caso de Prueba de Aceptación	
Código: HU_8_P1	**Historia de Usuario:** 8
Nombre: Guardar imagen del diseño	
Descripción: Prueba para la funcionalidad de guardar imagen del diseño	
Condiciones de Ejecución: La aplicación debe de estar en ejecución	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de exportar autómata como imagen	
Resultado Esperado: Que la herramienta permita seleccionar donde guardar la imagen del diseño y la guarde	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 56 Prueba de aceptación para la HU "Guardar el diseño de la máquina de estados en un archivo en formato XML"

Caso de Prueba de Aceptación	
Código: HU_9_P1	**Historia de Usuario:** 9
Nombre: Guardar el diseño de la máquina de estados	
Descripción: Prueba para la funcionalidad de guardar el diseño de la máquina de estados en formato XML	
Condiciones de Ejecución: La aplicación debe de estar en ejecución.	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de guardar • Se selecciona donde se quiere guardar y con qué nombre	
Resultado Esperado: Que la herramienta permita guardar el diseño de la máquina de estado en un archivo XML en la dirección especificada por el usuario	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 57 Prueba de aceptación para la HU "Cargar el diseño de la máquina de estados desde un archivo en formato XML"

Caso de Prueba de Aceptación	
Código: HU_10_P1	**Historia de Usuario:** 10
Nombre: Cargar el diseño de la máquina de estados desde un archivo	
Descripción: Cargar el diseño de la máquina de estados desde un archivo en formato XML	
Condiciones de Ejecución: La aplicación debe de estar en ejecución. Debe existir una máquina de estados guardada previamente con la herramienta.	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción de cargar • Se selecciona la máquina de estados guardada en XML	
Resultado Esperado: Que la herramienta permita cargar una máquina de estados guardada previamente desde cualquier dirección	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 58 Prueba de aceptación para la HU “Exportar a código C++”

Caso de Prueba de Aceptación	
Código: HU_11_P1	**Historia de Usuario:** 11
Nombre: Exportar a código C++	
Descripción: Prueba para la funcionalidad de exportar a código fuente C++	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y debe existir una máquina de estados en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona la opción exportar la máquina de estados a código fuente C++	
Resultado Esperado: Que la herramienta permita exportar la máquina de estados a código fuente C++ con la estructura que propone el patrón State	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 59 Prueba de aceptación para la HU “Probar el funcionamiento de un autómata”

Caso de Prueba de Aceptación	
Código: HU_12_P1	**Historia de Usuario:** 12
Nombre: Probar la máquina de estado	
Descripción: Prueba para la funcionalidad de probar el funcionamiento de un autómata	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y debe existir una máquina de estados en la escena	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Se selecciona un estado y se selecciona la opción de simular el funcionamiento de forma guiada	
Resultado Esperado: Que la herramienta muestre dos tablas, la primera contiene los estados por los que va pasando la máquina de estados y en la otra las transiciones del estado en que se encuentra y hacia qué estado va Debe dar la posibilidad al usuario de escoger que transición se ejecuta	
Evaluación de la Prueba: Prueba satisfactoria	

Tabla 60 Prueba de aceptación para la HU "Probar el funcionamiento de un autómata"

Caso de Prueba de Aceptación	
Código: HU_12_P2	**Historia de Usuario:** 12
Nombre: Probar la máquina de estado.	
Descripción: Prueba para la funcionalidad de probar el funcionamiento de un autómata.	
Condiciones de Ejecución: La aplicación debe de estar en ejecución y debe existir una máquina de estados en la escena.	
Entrada/Pasos de Ejecución: • Se ejecuta la aplicación • Debe el diseño debe tener definido el estado inicial y al menos un estado final. • Se selecciona la opción de probar de forma automática	
Resultado Esperado: • Que la aplicación muestre un mensaje que diga que debe selecciona el estado inicial de	

la máquina de estados • Que la aplicación muestre un mensaje que diga que debe existir un estado final en la máquina de estados • Que la aplicación muestre una ventana que muestre todos los caminos que no terminan en un estado final. • Que la aplicación muestre un mensaje que diga que el autómata funciona correctamente.
Evaluación de la Prueba: Prueba satisfactoria

Como parte del proceso de validación de la herramienta. Se llevó a cabo un experimento con una parte del equipo de desarrollo de los Laboratorios Virtuales. En una muestra de cuatro desarrolladores con experiencia entre los 2 y 5 años, se le pedía que crearan un máquina de estado desde el inicio sin utilizar la herramienta creada y utilizando la herramienta. La gráfica que se muestra a continuación tiene como objetivo comparar la variable tiempo, donde la escala de medida son los días que demora la creación de una máquina de estados. Se aprecia una disminución notable de los tiempos entre uno y otro caso.

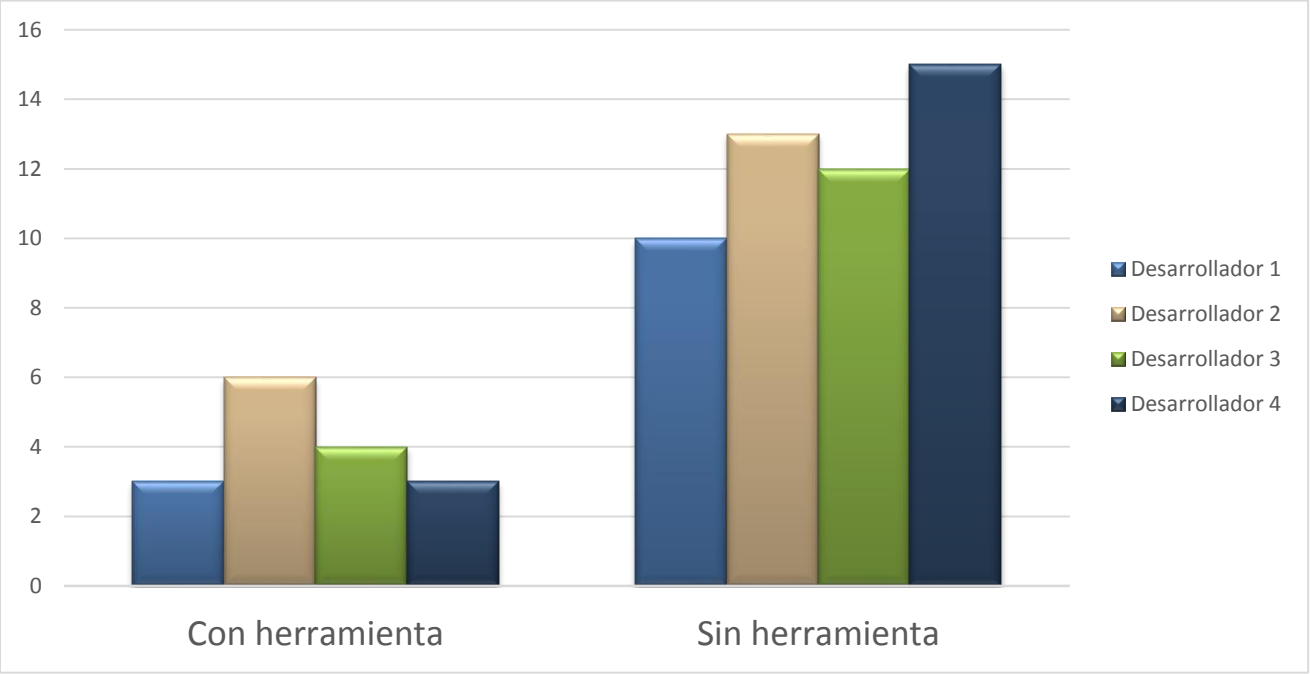


Figura 17 Gráfica donde se comparan los tiempos de desarrollo de una máquina de estados

Conclusiones Parciales

En este capítulo se realizó la implementación y validación de una versión de la herramienta propuesta que corresponde a 100% de la aplicación final. Se trataron las fases de construcción y prueba de la metodología de desarrollo XP. Se vieron las 4 iteraciones llevadas a cabo en la etapa de construcción de la aplicación, así como las tareas de programación correspondiente a cada historia de usuario. Se profundizó en el estudio de las pruebas unitarias y de aceptación, seleccionando las últimas como las más indicadas para comprobar el funcionamiento del software, debido a que demuestran la satisfacción del cliente con el producto.

CONCLUSIONES

Una vez finalizada la investigación se concluye que:

- El empleo de esta herramienta permite realizar cambios en el diseño del autómata y obtener los cambios correspondientes en el código de forma inmediata.
- Además permite detectar tempranamente errores en la concepción de la máquina de estados, que anteriormente solo se detectaban en la implementación.
- La aplicación de pruebas de aceptación posibilitaron validar los resultados de la investigación. Las pruebas demostraron que el software está listo para su uso.

La herramienta obtenida como resultado de la investigación da cumplimiento al objetivo planteado, permitiendo a los desarrolladores de los Laboratorios Virtuales una mejor gestión de autómatas.

RECOMENDACIONES

A la herramienta propuesta en la presente investigación, se le pueden aplicar un conjunto de modificaciones que mejoren su desempeño, por lo que se recomienda:

- Agregarle la funcionalidad de poder seleccionar entre múltiples idiomas para hacer más extensible la herramienta.
- Utilizar plantillas para que la modificación del estilo de codificación de las clases que se exportan sea más fácil, sin necesidad de intervenir en el código de la aplicación.

REFERENCIAS BIBLIOGRÁFICAS

1. P.Vary, James. ***Informe de la reunión de expertos.*** **Paris : s.n., 2000.**

2. COMPUMAT 2013. [En línea] [Citado el: 16 de marzo de 2014.] https://compumat.uci.cu/?q=node/947.

3. Hopcroft, John, Motwani, Rajeev y Ullman, Jeffrey. ***Introduction to automata theory, lenguages, and Computation.***

4. Unified Modeling Language™ (UML®). [En línea] Object Management Group, 28 de 05 de 2014. [Citado el: 2 de 6 de 2014.] http://www.uml.org/.

5. Mertz, David. deverloperWorks. ***Charming Python: Using state machines.*** **[En línea] IBM, agosto de 2000. http://www.ibm.com/developerworks/linux/library/l-python-state/index.html.**

6. State machine diagram. [En línea] [Citado el: 15 de febrero de 2014.] http://www.visual-paradigm.com/VPGallery/diagrams/State.html.

7. Design Patterns with UML. [En línea] 06 de febrero de 2013. http://design-patterns-with-uml.blogspot.com.ar/search/label/State%20Pattern.

8. CADIFRA. [En línea] Adrian & Frank Buehlmann. [Citado el: 15 de enero de 2014.] http://www.cadifra.com/features/.

9. Automata editor. [En línea] [Citado el: 15 de enero de 2014.] http://automataeditor.sourceforge.net/.

10. ALTOVA. ***UModel - UML tool for software modeling and application development.*** **[En línea] Altova. [Citado el: 16 de enero de 2014.] http://www.altova.com/umodel.html.**

11. ALTOVA. ***UML State Machine Diagrams.*** **[En línea] http://www.altova.com/umodel/state-diagrams.html.**

12. SPARX SYSTEMS. [En línea] [Citado el: 17 de enero de 2014.] http://www.sparxsystems.es/enterprisearchitect/.

13. Software design tools for agile software daverlopment. [En línea] [Citado el: 17 de enero de 2014.] http://www.visual-paradigm.com/.

14. [En línea] [Citado el: 17 de enero de 2014.] http://www.visual-paradigm.com/features/.

15. Google sites. TODO UML. [En línea] [Citado el: 28 de enero de 2014.] https://sites.google.com/site/todouml/herramientas/rational-rose.

16. IBM. ***Rational Rose Enterprise.*** **[En línea] [Citado el: 17 de enero de 2014.] http://www-03.ibm.com/software/products/es/enterprise.**

17. TODO UML. ***Rational Rose.*** **[En línea] Google sites. [Citado el: 17 de enero de 2014.] https://sites.google.com/site/todouml/herramientas/rational-rose.**

18. Aliander, Capdezuñer González y Carvajal Suárez, Adonis. ***Herramienta de configuración de estados y animaciones de un agente virtual autónomo para videojuegos 2D.*** **La Habana : s.n., 2010.**

19. cplusplus.com. [En línea] [Citado el: 4 de febrero de 2014.] http://www.cplusplus.com/info/description/.

20. W3C Ubiquitous Web domian. [En línea] W3C, 29 de 10 de 2013. [Citado el: 31 de enero de 2014.] http://www.w3.org/XML/.

21. Qt digia. [En línea] [Citado el: 5 de febrero de 2014.] http://qt.digia.com/Product/Qt-Framework/IDE--Tools/.

22. Extreme Programming: A gentle introduction. [En línea] 8 de october de 2013. http://www.extremeprogramming.org/.

23. Tecnología y Synergix. [En línea] 7 de julio de 2008. [Citado el: 8 de febrero de 2014.] http://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/.

24. Joskowicz, Ing.José. [En línea] 2008. http://iie.fing.edu.uy/~josej/docs/XP - Jose Joskowicz.pdf.

25. Gómez, Lic. Leopoldo Sebastián M. Monografias.com. [En línea] [Citado el: 24 de febrero de 2014.] http://www.monografias.com/trabajos10/diusuar/diusuar.shtml.

26. the java tutorial. [En línea] [Citado el: 1 de marzo de 2014.] http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html.

27. Larman, Craig. ***UML y PATRONES Una introduccion al analisis y diseño orientado a objetos y al proceso unificado.*** **págs. 356-378.**

28. Reynoso, Carlos y Kicillof, Nicolás. ***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*** **UNIVERSIDAD DE BUENOS AIRES. 2004. pág. 24.**

29. ARQUITECTURA EN CAPAS. [En línea] 10 de agosto de 2011. [Citado el: 5 de febrero de 2014.] http://arquitecturaencapas.blogspot.com/2011/08/arquitectura-3-capas-programacion-por.html.

30. Rao, Danya. GRASP Design Principles. [En línea] [Citado el: 25 de febrero de 2014.] http://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/rao.pdf.

31. Martínez, Ingeniero en Ciencias Informáticas Eduardo Salazar. Informática Jurídica.com. [En línea] [Citado el: 1 de abril de 2014.] http://www.informatica-juridica.com/trabajos/Propuesta_Procedimiento_para_realizar_pruebas_Caja_Blanca_aplicaciones_desarrollan_lenguaje_Python.asp.

BIBLIOGRAFÍA

1. P.Vary, James. ***Informe de la reunión de expertos.*** **Paris : s.n., 2000.**

2. COMPUMAT 2013. [En línea] [Citado el: 16 de marzo de 2014.] https://compumat.uci.cu/?q=node/947.

3. Hopcroft, John, Motwani, Rajeev y Ullman, Jeffrey. ***Introduction to automata theory, lenguages, and Computation.***

4. Unified Modeling Language™ (UML®). [En línea] Object Management Group, 28 de 05 de 2014. [Citado el: 2 de 6 de 2014.] http://www.uml.org/.

5. Mertz, David. deverloperWorks. ***Charming Python: Using state machines.*** **[En línea] IBM, agosto de 2000. http://www.ibm.com/developerworks/linux/library/l-python-state/index.html.**

6. State machine diagram. [En línea] [Citado el: 15 de febrero de 2014.] http://www.visual-paradigm.com/VPGallery/diagrams/State.html.

7. Design Patterns with UML. [En línea] 06 de febrero de 2013. http://design-patterns-with-uml.blogspot.com.ar/search/label/State%20Pattern.

8. CADIFRA. [En línea] Adrian & Frank Buehlmann. [Citado el: 15 de enero de 2014.] http://www.cadifra.com/features/.

9. Automata editor. [En línea] [Citado el: 15 de enero de 2014.] http://automataeditor.sourceforge.net/.

10. ALTOVA. ***UModel - UML tool for software modeling and application development.*** **[En línea] Altova. [Citado el: 16 de enero de 2014.] http://www.altova.com/umodel.html.**

11. ALTOVA. ***UML State Machine Diagrams.*** **[En línea] http://www.altova.com/umodel/state-diagrams.html.**

12. SPARX SYSTEMS. [En línea] [Citado el: 17 de enero de 2014.] http://www.sparxsystems.es/enterprisearchitect/.

13. Software design tools for agile software daverlopment. [En línea] [Citado el: 17 de enero de 2014.] http://www.visual-paradigm.com/.

14. [En línea] [Citado el: 17 de enero de 2014.] http://www.visual-paradigm.com/features/.

15. Google sites. TODO UML. [En línea] [Citado el: 28 de enero de 2014.] https://sites.google.com/site/todouml/herramientas/rational-rose.

16. IBM. ***Rational Rose Enterprise.*** **[En línea] [Citado el: 17 de enero de 2014.] http://www-03.ibm.com/software/products/es/enterprise.**

17. TODO UML. ***Rational Rose.*** **[En línea] Google sites. [Citado el: 17 de enero de 2014.] https://sites.google.com/site/todouml/herramientas/rational-rose.**

18. Aliander, Capdezuñer González y Carvajal Suárez, Adonis. ***Herramienta de configuración de estados y animaciones de un agente virtual autónomo para videojuegos 2D.*** **La Habana : s.n., 2010.**

19. cplusplus.com. [En línea] [Citado el: 4 de febrero de 2014.] http://www.cplusplus.com/info/description/.

20. W3C Ubiquitous Web domian. [En línea] W3C, 29 de 10 de 2013. [Citado el: 31 de enero de 2014.] http://www.w3.org/XML/.

21. Qt digia. [En línea] [Citado el: 5 de febrero de 2014.] http://qt.digia.com/Product/Qt-Framework/IDE--Tools/.

22. Extreme Programming: A gentle introduction. [En línea] 8 de october de 2013. http://www.extremeprogramming.org/.

23. Tecnología y Synergix. [En línea] 7 de julio de 2008. [Citado el: 8 de febrero de 2014.] http://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/.

24. Joskowicz, Ing.José. [En línea] 2008. http://iie.fing.edu.uy/~josej/docs/XP - Jose Joskowicz.pdf.

25. Gómez, Lic. Leopoldo Sebastián M. Monografias.com. [En línea] [Citado el: 24 de febrero de 2014.] http://www.monografias.com/trabajos10/diusuar/diusuar.shtml.

26. the java tutorial. [En línea] [Citado el: 1 de marzo de 2014.] http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html.

27. Larman, Craig. ***UML y PATRONES Una introduccion al analisis y diseño orientado a objetos y al proceso unificado.*** **págs. 356-378.**

28. Reynoso, Carlos y Kicillof, Nicolás. ***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*** **UNIVERSIDAD DE BUENOS AIRES. 2004. pág. 24.**

29. ARQUITECTURA EN CAPAS. [En línea] 10 de agosto de 2011. [Citado el: 5 de febrero de 2014.] http://arquitecturaencapas.blogspot.com/2011/08/arquitectura-3-capas-programacion-por.html.

30. Rao, Danya. GRASP Design Principles. [En línea] [Citado el: 25 de febrero de 2014.] http://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/rao.pdf.

31. Martínez, Ingeniero en Ciencias Informáticas Eduardo Salazar. Informática Jurídica.com. [En línea] [Citado el: 1 de abril de 2014.] http://www.informatica-juridica.com/trabajos/Propuesta_Procedimiento_para_realizar_pruebas_Caja_Blanca_aplicaciones_desarrollan_lenguaje_Python.asp.

32. Visconti, Marcello y Astudillo, Hernan. Fundamentos de Ingenieria de software. [En línea] [Citado el: 15 de febrero de 2014.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

33. Fani Calle. [En línea] 24 de junio de 2008. http://www.slideshare.net/Decimo/arquitectura-3-capas.

34. Zhang, Kevin. ***Design Patterns Elements of Reusable Object-Oriented Software.***

35. Facultad de informática - Universidad Politécnica de Madrid. ***Patrones del "Gang of Four".***

36. UML 2 State Machine Diagrams: An Agile Introduction. [En línea] [Citado el: 15 de enero de 2014.] http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm.

37. PROGRAMACIÓN EXTREMA (XP). [En línea] [Citado el: 8 de marzo de 2014.] http://ingenieriadesoftware.mex.tl/images/18149/PROGRAMACIÓN EXTREMA.pdf.

38. Gutiérrez, J. J., Escalona, M. J. y M. Mejías, J. Torres. PRUEBAS DEL SISTEMA EN PROGRAMACIÓN. [En línea] [Citado el: 12 de marzo de 2014.] http://www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf.

39. Google Sites. Metodología XP. [En línea] [Citado el: 8 de marzo de 2014.] https://sites.google.com/site/xpmetodologia/home/introduccion.

40. —Jummp Gestión de proyectos y desarrollo de software. [En línea] 10 de enero de 2012. http://jummp.wordpress.com/2012/01/10/desarrollo-de-software-tarjetas-crc/.

41. Fases de la Programación Extrema. [En línea] [Citado el: 29 de enero de 2014.] http://programacionextrema.tripod.com/fases.htm.

42. Villegas, Adrian Anaya. Monografias.com. ***A propósito de programación extrema XP (eXtreme Programming).*** **[En línea] [Citado el: 16 de febrero de 2014.] http://www.monografias.com/trabajos51/programacion-extrema/programacion-extrema.shtml.**

Model-Driven Architecture: Arquitectura dirigida por modelos (**Model-Driven, Architecture** o MDA) es una arquitectura que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos.

Java: lenguaje de programación originalmente desarrollado por Sun Microsystems, adquirida por Oracle, para aplicaciones software independiente de la plataforma,

C#: es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común

Visual Basic.net: Visual Basic .NET (VB.NET) es un lenguaje de programación orientado a objetos que se puede considerar una evolución de Visual Basic implementada sobre el *framework* .NET.

HTML: siglas de *HyperText Markup Language* («lenguaje de marcas de hipertexto»), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones.

C++: es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos.

Python: **Python** es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

PHP: es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

PDF: (sigla del inglés *portable document format*, formato de documento portátil) es un formato de almacenamiento de documentos digitales independiente de plataformas de software o hardware.

JPG: También conocido como JPEG(del ingles *Joint Photographic Experts Group*, Grupo Conjunto de Expertos en Fotografía) es el nombre de un comité de expertos que creó un estándar de compresión y codificación de archivos e imágenes fijas.

PNG: (siglas en inglés de *Gráficos de Red Portátiles*, pronunciadas *"ping"*) es un formato gráfico basado en un algoritmo de compresión sin pérdida para bitmaps no sujeto a patentes.

ANEXOS

Tabla 61 Comparación entre herramientas

Herramienta	Características						
	Exportar en XML	**Importar en XML**	**Guardar como imagen**	**Deshacer y rehacer**	**Probar el funcionamiento**	**Exportar a código**	**Multiplataforma**
Cadifra UML Editor	Si	Si	Si	Si	No	No	Si
Automata Editor	No	No	Si	Si	Si	No	No
Altova Umodel 2014	Si	Si	Si	Si	No	Si	No
Enterprise Architect 7.0	Si	Si	Si	Si	Si	Si	No
Visual Paradim	Si	Si	Si	Si	No	Si	Si
Rational Rose	Si	Si	Si	Si	No	Si	Si
HCEA	Si	Si	No	Si	No	Si	Si

Estructura del archivo XML

```
<MACHINE>
    <STATES>
        <STATE>
            <VALUE></VALUE>
            <X></X>
            <Y></Y>
            <TEXT></TEXT>
            <INITIAL></INITIAL>
            <END> </END>
            <BRUSH></BRUSH>
            <FONT>
                <TYPE></TYPE>
                <SIZE></SIZE>
                <BOLD></BOLD>
                <ITALIC></ITALIC>
                <UNDERLINE></UNDERLINE>
                <STRIKEOUT></STRIKEOUT>
                <KERNING></KERNING>
            </FONT>
            <PEN>
                <COLOR></COLOR>
                <WIDTH></WIDTH>
                <STYLE></STYLE>
            </PEN>
            <ATTRIBUTES>
                <ATTRIBUTE>
                    <NAME></NAME>
                    <TYPE></TYPE>
                    <VISIBILITY></VISIBILITY>
                    <INITIAL_VALUE></INITIAL_VALUE>
                </ATTRIBUTE>
```

```
                </ATTRIBUTES>
                <FUNTIONS>
                        <FUNTION>
                                <NAME></NAME>
                                <TYPE></TYPE>
                                <VISIBILITY></VISIBILITY>
                                <PARAMETERS>
                                        <PARAMETERS>
                                                <NAME></NAME>
                                                <TYPE></TYPE>
                                                <INITIAL_VALUE></INITIAL_VALUE>
                                        </PARAMETERS>
                                </PARAMETERS>
                        </FUNTION>
                </FUNTIONS>
        </STATE>
</STATES>
<TRANSITIONS>
        <TRANSITION>
                        <NOMBRE></NOMBRE>
                        <START></START>
                        <END></END>
                <FONT>
                        <TYPE></TYPE>
                        <SIZE></SIZE>
                        <BOLD></BOLD>
                        <ITALIC></ITALIC>
                        <UNDERLINE></UNDERLINE>
                        <STRIKEOUT></STRIKEOUT>
                        <KERNING></KERNING>
                </FONT>
                <PEN>
                        <COLOR></COLOR>
```

```
                    <WIDTH></WIDTH>
                    <STYLE></STYLE>
                </PEN>
            </TRANSITION>
        </TRANSITIONS>
</MACHINE>
```

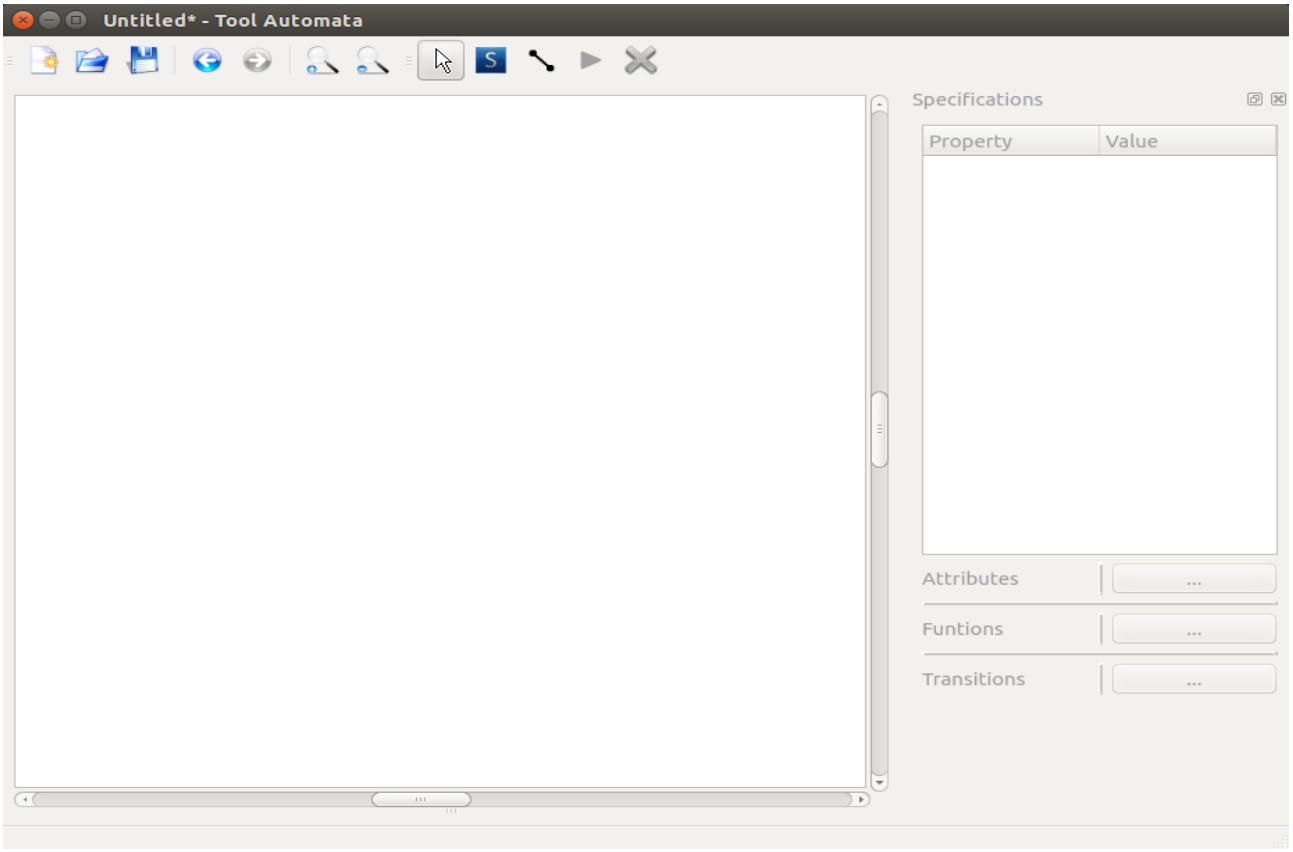


Figura 18 Interfaz principal

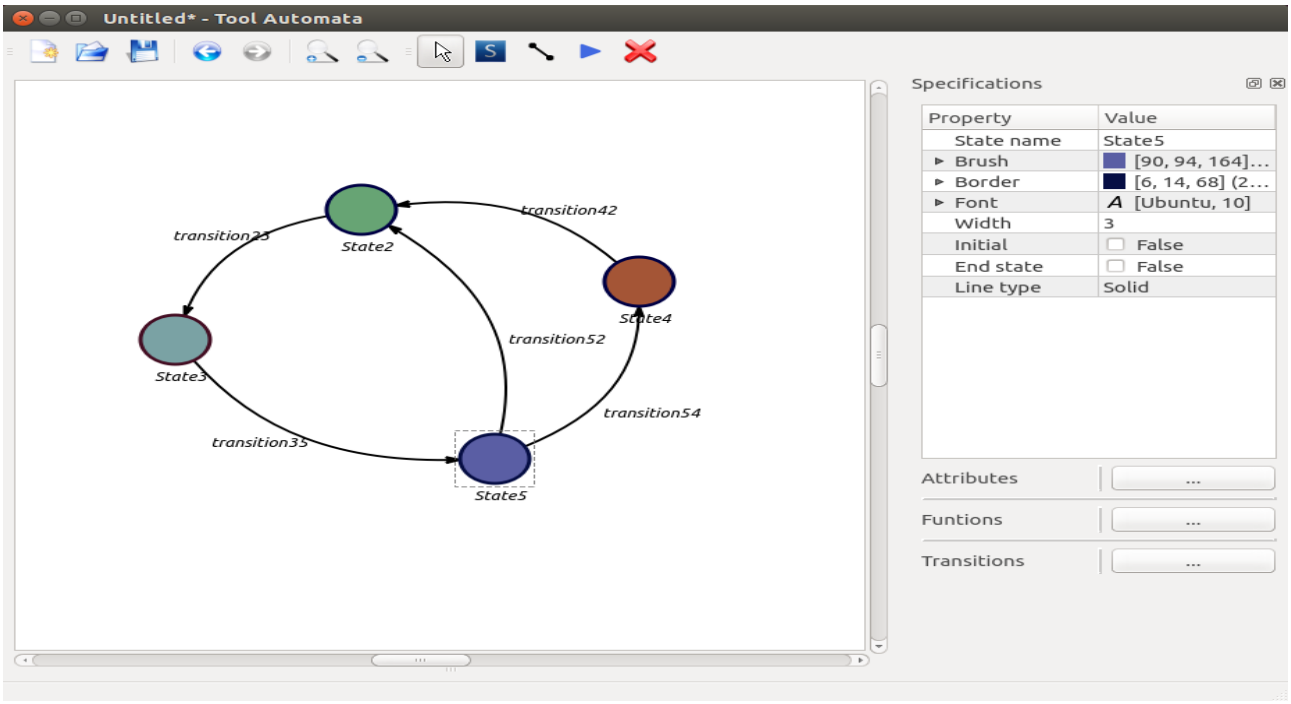


Figura 19 Ventana de gestión de estados y transiciones.

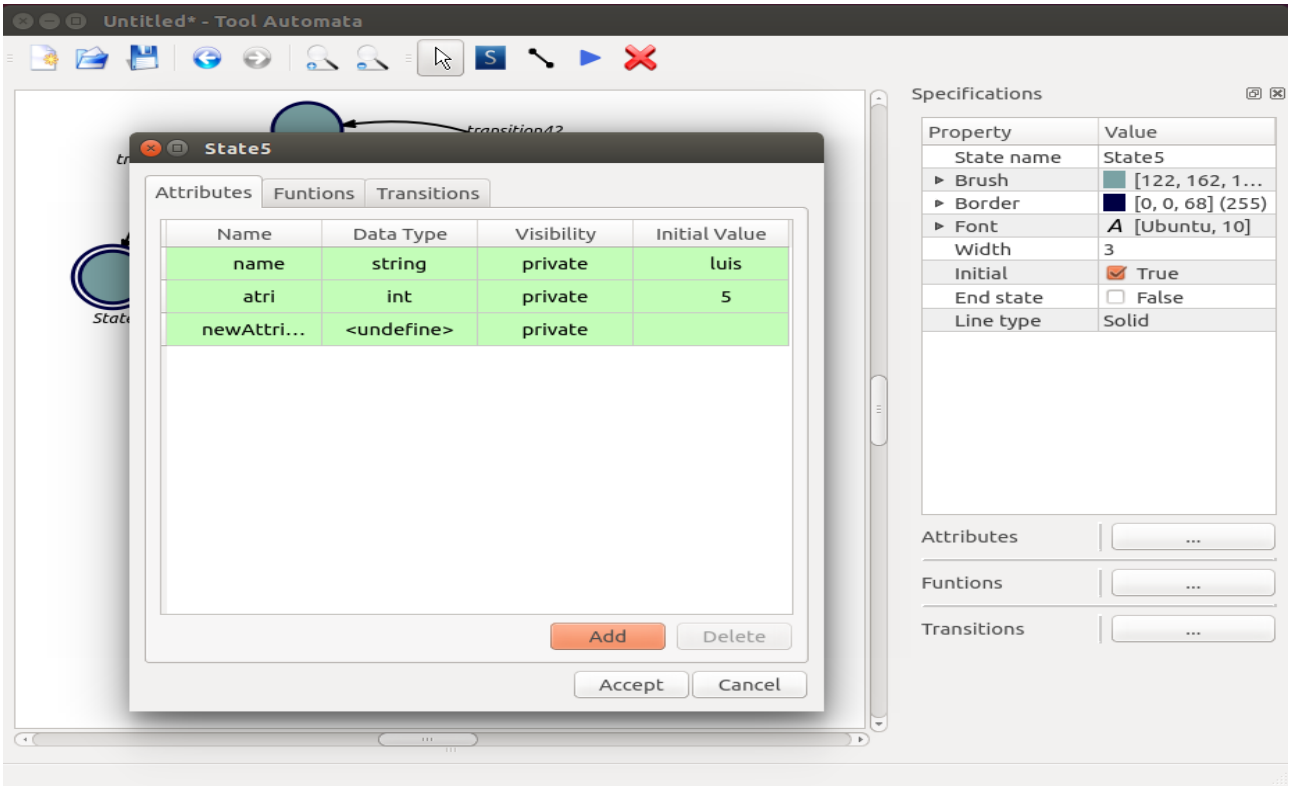


Figura 20 Ventana de gestión de atributos de un estado.

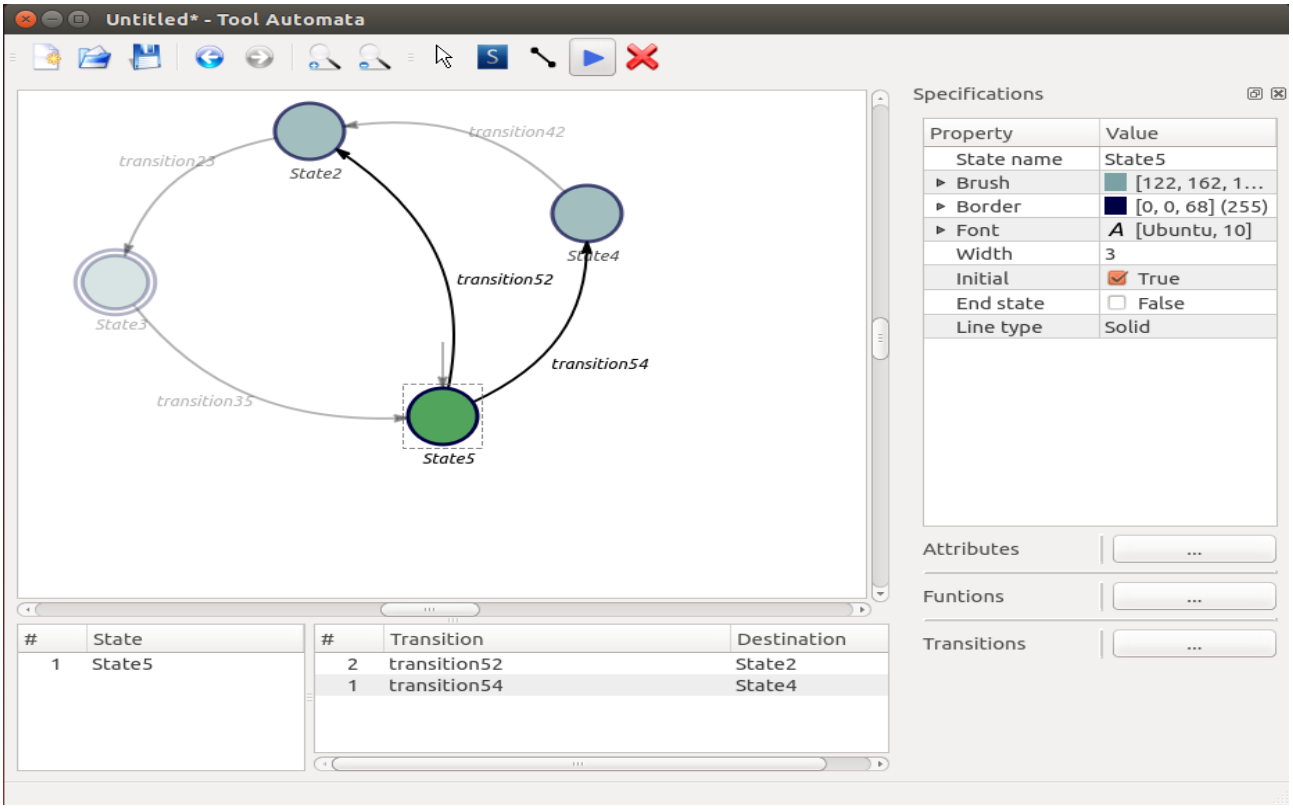


Figura 21 Imagen de la vista de probar de forma guiada.