UNIVERSIDAD NACIONAL DE COLOMBIA

Pruebas de Software Basadas en Modelos aplicadas en la Generación Automatizada de Casos de Prueba sobre Interfaces Gráficas de Usuario

Andrés Leonardo Cubillos Rodríguez

Universidad Nacional de Colombia

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación

Bogotá – Colombia

2012

Pruebas de Software Basadas en Modelos aplicadas en la Generación Automatizada de Casos de Prueba sobre Interfaces Gráficas de Usuario

Andrés Leonardo Cubillos Rodríguez

Trabajo Final de Maestría presentado como requisito parcial para optar al título de:

Magister en Ingeniería de Sistemas y Computación

Director:

Magister Henry Roberto Umaña Acosta

Plan de Estudios de Profundización.

Línea de Profundización:

Model Based Testing (*Pruebas Basadas en Modelos*)

Grupo de Investigación:

Ingeniería de Software – ColSWE

Universidad Nacional de Colombia

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación

Bogotá – Colombia

2012

Este trabajo está dedicado a mis padres y hermanos quienes me han brindado todo su apoyo, compresión y han sido fuente de inspiración para alcanzar las metas que me he propuesto.

Agradecimientos

Agradezco principalmente al Ingeniero Henry Roberto Umaña Acosta asociado al Departamento de Ingeniería de Sistemas e Industrial de la Universidad Nacional de Colombia quien fue el director y guía metodológico de este Trabajo Final de Maestría. Aportó su asesoría, apoyo, experiencia y conocimientos para poder realizar el proyecto y encaminó la estructuración del documento.

Finalmente agradezco a mi familia y amigos por su constante apoyo y compresión para poder lograr las metas propuestas en este trabajo.

Resumen

Este trabajo presenta una aplicación de las pruebas basadas en modelos (*Model Based Testing - MBT*) para automatizar las pruebas sobre interfaces gráficas de usuario (*Graphical User Interfaces - GUI*) de uso común en aplicaciones de escritorio (como formularios y vistas compuestas) empleando la herramienta *Spec Explorer 2010* de Microsoft. Se definen los aspectos fundamentales que se deben modelar de la GUI y cómo se debe construir el modelo a partir del cual se generan los casos de prueba, los datos de entrada de pruebas y los resultados esperados. Para implementar MBT sobre una GUI se propone una arquitectura basada en el patrón MVP (*Model View Presenter*), que se debe aplicar tanto en la fase de diseño del modelo como en la fase de implementación, para lograr el comportamiento deseado de la interfaz de acuerdo a los escenarios de los casos de uso. Aunque la generación de los casos de prueba con el enfoque propuesto implica tiempo y esfuerzo inicial, brinda ventajas como ser un proceso efectivo en implementar el comportamiento deseado de la GUI y de buena cobertura de pruebas.

Palabras clave: Automatización de casos de pruebas, pruebas dirigidas por modelos, pruebas sobre interfaces gráficas de usuario, patón modelo-vista-presentador, Spec Explorer 2010.

Abstract

This work shows an application of the concepts of model-based testing to automate testing through graphical user interfaces (GUI) used in desktop applications, mainly through forms. The MBT tool used was Spec Explorer 2010 from Microsoft. The main features of the graphical user interface are defined and then modeled to generate test cases, related data and the result of the oracle. The architecture chosen was based on MVP pattern (Model View Presenter, which should be applied in two phases: model design and implementation in accordance with the expected behavior defined in use cases. Using this technique involves a lot of effort and time, but this situation has some advantages such as increased test coverage and being a testing process effective to implement the GUI's wanted behavior.

Key Words: Model-based testing, Spec Explorer 2012, testing from GUIs, tests automated, model view presenter pattern.

Contenido

Lista de figuras

Lista de tablas

1. Introducción

En todo proceso de desarrollo de software existe una fase de pruebas del producto o sistema antes de que sea desplegado en producción y pueda ser empleado directamente por el usuario final. El objetivo de este proceso de pruebas es asegurar la calidad, confiabilidad y robustez del software, dentro de un ambiente o contexto en el cuál será empleado [3].

Como parte de las pruebas de software, se encuentran las que aseguran el correcto funcionamiento de las interfaces gráficas de usuario (GUIs - *Graphical User Interfaces*). Ya que la interfaz gráfica de una aplicación es por lo general el medio con el cual el usuario interaccionará con el sistema; las pruebas de este tipo son trascendentales para que un software sea aceptado por el usuario final y sea desplegado en producción [3], además es el punto de entrada para que un cliente perciba que efectivamente se están satisfaciendo sus necesidades.

En la actualidad existen herramientas y técnicas para la automatización de pruebas de software sobre interfaces graficas de usuario [18] , pero por lo general estos procesos son costosos, difíciles de mantener, poco flexibles a cambios de requerimientos, no mantienen una buena trazabilidad y su cobertura es imprecisa para probar todas las funcionalidades [1].

El proceso de pruebas basadas en modelos (*Model Based Testing- MBT*) soluciona los inconvenientes señalados anteriormente [1]. En esta técnica, se empieza por delimitar el Sistema Bajo Prueba (*System Under Test - SUT*) o las partes del sistema que van a ser probadas. Después de esto, empleando alguna técnica existente (basada en transiciones o basada en pre y pos condiciones), se modela el SUT. A partir de este modelo se generan casos de prueba abstractos o de alto nivel, los cuales están en términos del modelo y no en términos de un lenguaje específico. Para ejecutar los casos de prueba

abstractos en una plataforma en particular, se concretizan, es decir, se transforman en scripts o código, de manera que puedan ser ejecutados sobre el SUT. Por último, se analizan los resultados (pruebas que pasan y pruebas que fallan), y se elabora el reporte correspondiente.

Aprovechando las ventajas brindadas por el proceso MBT, se propone aplicarlo para automatizar la generación de casos de prueba sobre interfaces gráficas de usuario. Las interfaces gráficas, que se utilizarán como objetivo para las pruebas, son las de uso común en aplicaciones de escritorio[1], como formularios o vistas compuestas. Para poder conseguir esto, primero se definen los aspectos que se pueden modelar de una interfaz gráfica de usuario específica, y luego se modela su comportamiento diseñando un modelo a partir del cual se generen los casos de prueba, los datos de entrada de pruebas y los resultados esperados.

Para poder implementar MBT sobre GUIs, se propuso una arquitectura basada en el patrón MVP (*Model View Presenter- Modelo Vista Presentador* [25]), que debe ser implementada tanto en el modelo como en la implementación para lograr el comportamiento deseado de la interfaz de acuerdo a los requerimientos. Con este planteamiento, se evita depender de la tecnología concreta de la interfaz gráfica de usuario y del conocimiento previo de los detalles internos de su implementación, pues se aprovecha las características principales de adoptar este patrón, como son la abstracción y la reutilización [25, 29, 30]. Para el proyecto se trabajó con *Windows Forms,* que es una tecnología de Microsoft para diseñar interfaces gráficas de usuario de aplicaciones de escritorio.

Para los casos de ejemplo propuestos se obtuvo una cobertura promedio de código del 90%, pues la implementación se desarrollaba enfocándose en pasar las pruebas generadas previamente a partir del modelo.

[1] Una aplicación de escritorio es aquella que se instala en la máquina local del usuario, se ejecuta directamente por el sistema operativo y su rendimiento está directamente relacionado con los recursos de hardware del equipo. Las GUIs de estas aplicaciones contienen una estructura jerárquica de elementos gráficos que caracterizan su estado, y la interacción con el usuario es dirigida por eventos [10].

La construcción del modelo de pruebas implicó tiempo y esfuerzo inicial, sin embargo brindó ventajas como ser un proceso efectivo en implementar el comportamiento deseado del sistema de acuerdo a los requerimientos y de tener buena cobertura de pruebas. Al final del proceso se generaron más de 300 casos de prueba sobre una interfaz gráfica de usuario compuesta.

1.1 Alcances y Limitaciones

Este trabajo está orientado a la aplicación de MBT sobre GUIs, empleando herramientas y técnicas de modelado ya existentes; no pretende la elaboración de una herramienta completa para el desarrollo del proceso de pruebas basadas en modelos, la evaluación y pruebas de concepto de herramientas MBT, ni la definición de una metodología que sea implementada en un equipo de desarrollo de software. Además, no se aplicará el proceso MBT para probar la interfaz gráfica de usuario o capa de presentación completa de una aplicación específica**,** sino que se plantearán algunos casos de ejemplo con interfaces simples y complejas, como formularios y vistas compuestas, de uso común en aplicaciones de escritorio. En el trabajo se utiliza la herramienta *Spec Explorer 2010* de Microsoft [15], con la cual se puede implementar todo el proceso de MBT sobre un sistema determinado.

1.2 Objetivos

1.2.1 General

El objetivo principal del proyecto es automatizar la generación de casos de prueba para interfaces gráficas de usuario de uso común en aplicaciones de escritorio (como formularios y vistas compuestas), aplicando el proceso de pruebas basadas en modelos.

1.2.2 Específicos

- ✓ Definir los aspectos que se pueden modelar de una interfaz gráfica de usuario para poder emplear el proceso de pruebas basadas en modelos.
- ✓ Emplear una técnica existente en las pruebas basadas en modelos para modelar el comportamiento de una interfaz gráfica de usuario específica (como formularios y vistas compuestas).

- ✓ Construir un modelo a partir del cual se genere los casos de prueba, los datos de entrada de pruebas y los resultados esperados.
- ✓ Analizar los resultados de la técnica empleada, respecto a su cobertura y efectividad.

1.3 Organización del Documento

Este trabajo está organizado de la siguiente manera: en el capítulo 2, se presenta una breve descripción de los métodos tradicionales empleados para realizar pruebas automáticas, y se contextualiza sobre el proceso de pruebas basadas en modelos (MBT). En el capítulo 3, se describe la manera como se implementó el proceso de pruebas basadas en modelos para automatizar los casos de pruebas sobre las GUIs objetivo. Para lograr esto se presenta la definición de los aspectos del comportamiento a modelar, el diseño del modelo concreto de pruebas, la automatización de los casos de prueba y la arquitectura que debe implementar la GUI para acoplarse con el comportamiento definido en el modelo. En el capítulo 4, se muestra el análisis de los resultados obtenidos y en el capítulo 5, se presentan las conclusiones y recomendaciones para un trabajo futuro. Finalmente en los Anexos se presenta la descripción de los casos de ejemplo realizados para dar soporte al enfoque planteado en este trabajo.

2. Contextualización

Todos los modelos de desarrollo de software (modelo de cascada, el modelo de espiral, el modelo evolutivo, el proceso unificado, el modelo V, el modelo W, entre otros) [17] contemplan un proceso de pruebas de software. En [21] y [28] se presenta una recopilación de los diferentes niveles de prueba, los cuales se resumen en la Tabla 1-1.

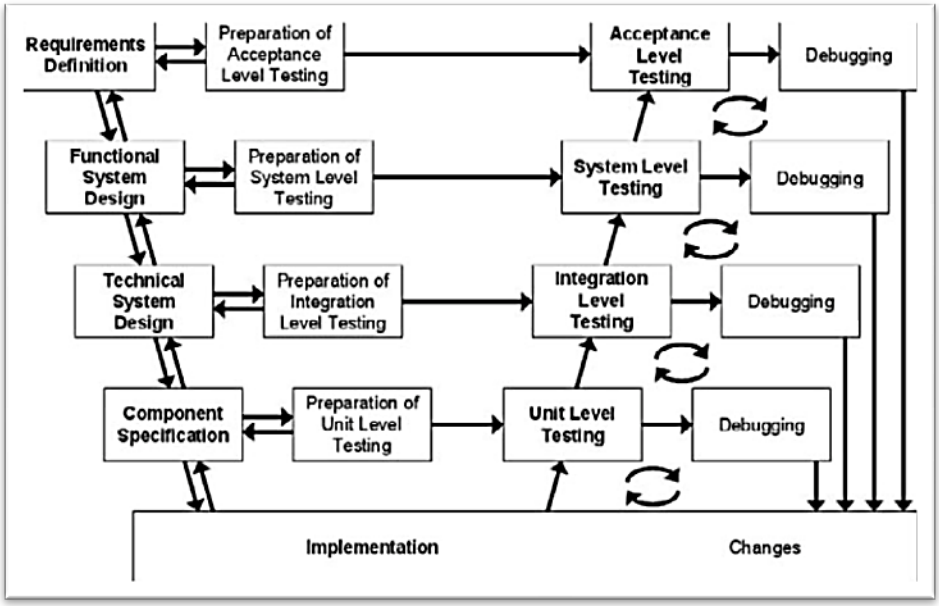
Tabla 1-1: Niveles de Prueba de Software [21].

Nivel de Prueba	Descripción
Unitarias	Prueban la unidad más pequeña del diseño del software: los métodos o funciones de una clase.
Integración	Prueban la correcta relación entre los componentes del sistema a través de sus interfaces y si ellas cumplen con la funcionalidad establecida.
Sistema	Prueban el sistema comprobando su funcionalidad y atributos de calidad. El sistema es probado en un ambiente lo más parecido posible al ambiente operacional.
Aceptación	Evalúan que el sistema cumple con todos los requisitos indicados y permite que los usuarios del sistema provean su aceptación.
Regresión	El objetivo es comprobar que los cambios sobre un componente del sistema no generan errores adicionales en otros componentes no modificados.

Un proceso comúnmente utilizado es el modelo W [17], ilustrado en la Figura 1-1. En el lado izquierdo del modelo las fases de construcción están estructuradas en dos conjuntos de tareas: las tareas relacionadas con las fases de desarrollo y las actividades de preparación (plan de pruebas) para cada fase correspondiente. La particularidad del modelo es la detección y corrección iterativa de errores, es decir, si una prueba detecta

un fallo entonces debe ser localizado, una vez es localizado se procede a su corrección, lo que significa un cambio en la implementación del software. Al ser cambiada la implementación la prueba tiene que ser ejecutada nuevamente.

Figura 1-1: El Modelo W de Desarrollo de Software [17].



2.1 Técnicas Tradicionales de Automatización de Pruebas

Ya que el desarrollo de pruebas es una actividad diseñada para analizar la calidad de un producto mediante un conjunto finito de casos de pruebas, este puede ser automatizado [1]. A continuación se mencionarán brevemente los métodos tradicionales de automatización de pruebas de software.

2.1.1 Capture & Replay Testing

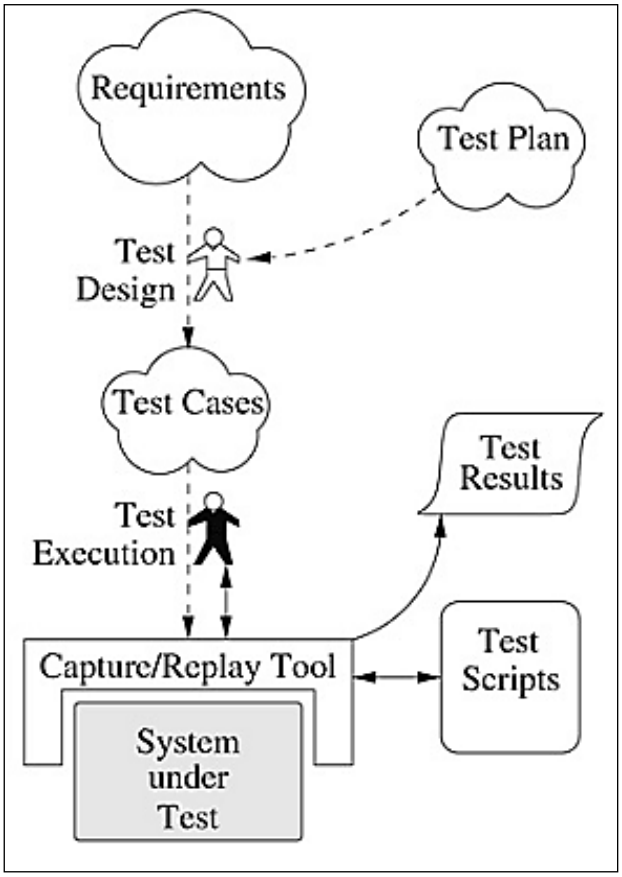
En este tipo de pruebas los casos de prueba se generan de forma manual, pero se capturan las secuencias durante una sesión de ejecución del conjunto de pruebas, posteriormente se reproducen estas interacciones de forma automática en el SUT (System Under Test) [1, 18].

Una herramienta de este tipo registra todas las entradas enviadas al sistema bajo prueba y los resultados obtenidos (retorno de procedimientos, resultados de funciones, capturas

de pantalla, archivos de datos, entre otras), de manera que cuando una nueva versión deba ser probada, la herramienta de captura y reproducción realizará una nueva ejecución de todas las pruebas grabadas. Al ejecutar cada prueba grabada, se analizan las entradas registradas y se comparan los resultados con los obtenidos anteriormente.

El problema con esta clase de pruebas es que no son muy robustas, si se realiza una pequeña actualización sobre la interfaz gráfica, las grabaciones anteriores no servirán y se tendrán que generar nuevas [1]. El proceso general de pruebas capture/replay es mostrado en la Figura 1-2:

Figura 1-2: Proceso de Pruebas Capture & Replay [1].



La reproducción de capturas no es un modelo de automatización realmente, la reproducción se inicia luego de la prueba manual y la verificación se debe realizar también de forma manual. No es lo mismo una prueba automática a probar automáticamente. Probar automáticamente no solo significa ejecutar pruebas de manera

automática, sino también generar las pruebas, los datos de entrada y los resultados esperados de manera automatizada [1].

2.1.2 Scripting Testing

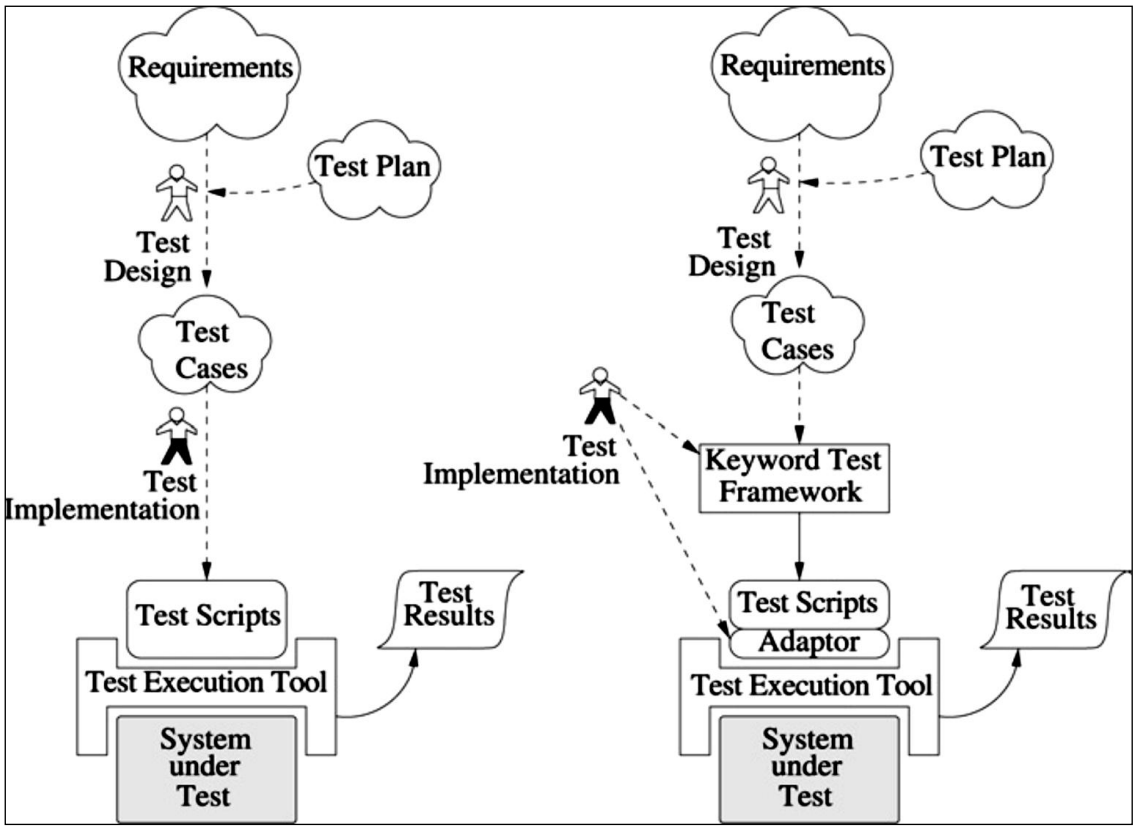
En este tipo de pruebas se utilizan scripts ejecutables para la realización de las pruebas, mediante dichos scripts se pueden ejecutar varios casos de pruebas a la vez, se generan varios valores de entrada y se verifican las salidas apoyándose de alguna API para el diseño de los mismos.

En este proceso se aumenta el problema del mantenimiento del set de pruebas ya que éste puede evolucionar no sólo por el cambio de algunos requerimientos, sino también cada vez que algunos detalles de implementación cambien, dado que el tamaño total del set de pruebas puede ser casi tan grande como la aplicación bajo prueba, con esto los scripts de prueba son muy costosos de mantener [18].

La abstracción es importante para la solución de este problema [1], haciendo los scripts de ejecución de los casos de prueba mucho más genéricos y de alto nivel, pero la abstracción depende de la habilidad del diseñador y del dominio que tenga en la programación de la API respectiva para la generación del código y finalmente la construcción de los ejecutables pertinentes. Sin embargo los datos de prueba y la verificación de acuerdo a los requisitos todavía se tienen que hacer de forma manual.

A continuación se presenta el proceso de pruebas basados en scripts (Figura 1-3), a la izquierda se muestra el proceso básico y a la derecha el proceso evolucionado que maneja un grado de abstracción al separar los scripts de prueba de alto nivel con la ejecución final.

Figura 1-3: Proceso de Pruebas Basados en Scripts [1].



En la Tabla 1-2 se presenta un consolidado de las diferentes técnicas y métodos empleados en un proceso de pruebas de software, el nivel indica la robustez del método en cuanto a automatización, costo de mantenimiento, flexibilidad, cobertura, trazabilidad, capacidad de regresión y generación de resultados esperados; para cada una se exponen sus alcances y las falencias principales. Podemos observar que el proceso de pruebas basadas en modelos es el más robusto.

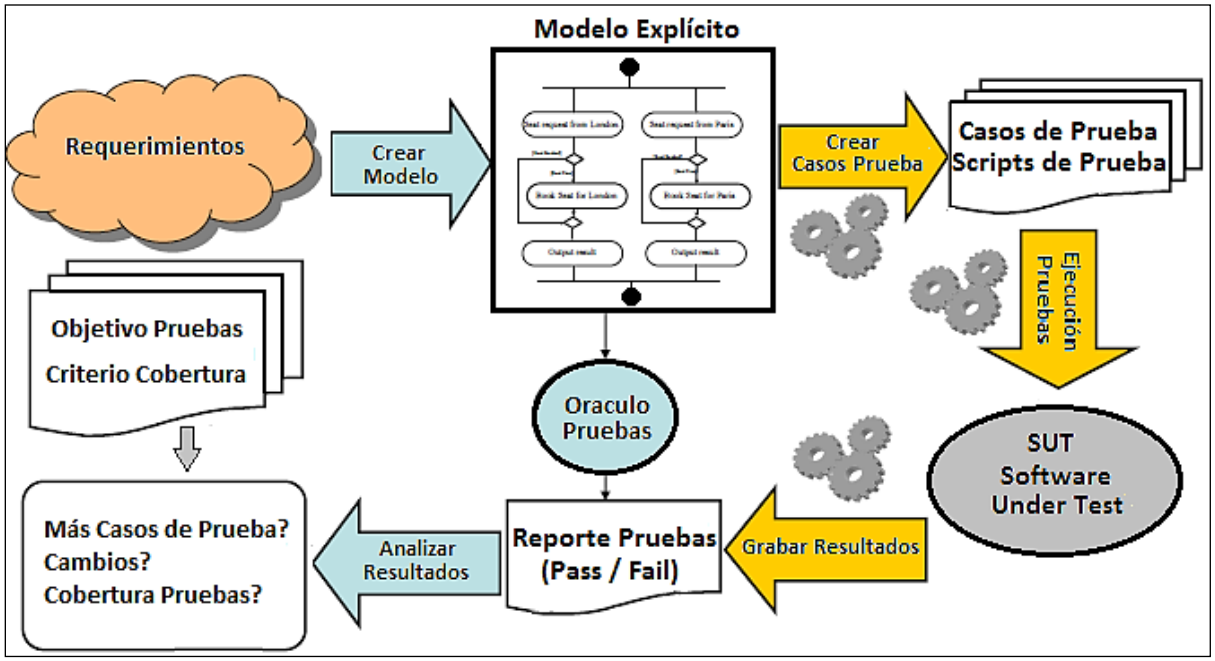
Tabla 1-2: Métodos empleados en las Pruebas de Software [1, 18].

Nivel	Pruebas	Alcance	Falencias
1	Manuales	Se enfoca en las pruebas funcionales	Cobertura es imprecisa para probar todas las funcionalidades.
			Carencia en la implementación de pruebas de Regresión.
			Proceso muy costoso y lento.
			No existe una medida efectiva de la cobertura de las pruebas.
2	Capture – Replay	Hace lo posible para ejecutar automáticamente los casos de pruebas capturados.	Cobertura es imprecisa para probar todas las funcionalidades.
			Debilidad en la capacidad de realizar pruebas de regresión.
			Sensible a los cambios de la Interfaz Gráfica de Usuario.
			Cada cambio involucra re-capturar manualmente casos de prueba.
3	Basadas en Scripts	Hace lo posible para ejecutar y re-ejecutar automáticamente los scripts de prueba.	Cobertura es imprecisa para probar todas las funcionalidades.
			Los Scripts pueden llegar a ser muy complejos y difíciles de mantener.
			La trazabilidad de los requerimientos es desarrollada manualmente.
4	Pruebas dirigidas por Datos y “KeyWords”	Desarrollo de Scripts con un nivel más alto de abstracción, para disminuir el costo de mantenimiento.	Cobertura es imprecisa para probar todas las funcionalidades.
			La trazabilidad de los requerimientos es desarrollada manualmente.
5	Pruebas Basadas en Modelos	Automatización en la generación de casos de pruebas funcionales.	Dependencia inicial en la experiencia de los diseñadores de pruebas.
		Generación de los resultados esperados.	
		Reducción en el costo de mantenimiento.	
		Generación Automática de la matriz de trazabilidad de requerimientos.	

2.2 Model Based Testing (MBT)

Como se mencionó en la sección anterior uno de los niveles más robustos para automatizar las pruebas de software es el proceso de pruebas basadas en modelos (Model-Based Testing, MBT [1], [2], [5], [7], [8]). Este método tiene como objetivo generar automáticamente casos de pruebas ejecutables basándose en el modelo y diseño de alto nivel del sistema o de la aplicación.

Figura 1-4: Proceso de Pruebas de Software Basadas en Modelos [13].



El proceso de pruebas basadas en modelos es mostrado en la Figura 1-4, el cual es compuesto de los siguientes pasos fundamentales [1], [2], [5] :

- ✓ Construir un modelo que abstraiga los aspectos más importantes del comportamiento del sistema, basándose en los requerimientos del SUT.
- ✓ Generar casos de pruebas abstractos a partir del modelo. Estas pruebas son de alto nivel, pues son una representación lógica de los casos de prueba y no están descritas en un lenguaje o tecnología en particular.
- ✓ Concretizar los casos de pruebas abstractos. Las pruebas de alto nivel se concretizan en scripts o código de alguna tecnología específica para que sean ejecutables.

- ✓ Ejecutar los casos de pruebas concretos sobre el SUT.
- ✓ Obtener el reporte de resultados de ejecución de la suite de pruebas.
- ✓ Analizar los resultados de la suite de pruebas.

MBT implementado en un proceso de desarrollo de software conlleva los siguientes beneficios [1]:

- ✓ Detección de Errores en el Sistema Bajo Prueba.
- ✓ Reducción del Tiempo y Costo del Proceso de Pruebas.
- ✓ Mejora en la Calidad de Pruebas.
- ✓ Detección de Requerimientos Defectuosos.
- ✓ Trazabilidad.
- ✓ Evolución de Requerimientos.

2.2.1 Técnicas de Construcción de Modelos de Pruebas

Las técnicas más comunes empleadas para la creación del modelo en el proceso de pruebas basadas en modelos son desplegadas en la Tabla 1-3. Estas son notaciones que ayudan a diseñar el modelo del software bajo prueba y son el punto de partida para generar los casos de pruebas.

Tabla 1-3: Técnicas empleadas para el Modelado de Pruebas [1, 4, 9, 10, 12, 14, 17].

Técnicas Para Modelado	
Basadas en Pre/Post Condiciones	Notación B
	Object Constarin Language (OCL)
	Java Modeling Language
	Spec #
	VDM
	Notación Z
Basadas en Transición	Máquina de Estados Finito (FSM)
	Diagrama de Flujo de Eventos (Event-Flow Graph)
Basadas en UML	StateChart (UML State Machines)
	UML Testing Profile.

A continuación se mencionarán brevemente los aspectos más importantes de las técnicas de modelado más comunes en el proceso MBT.

2.2.1.1 Modelos Basados en Pre – Pos condiciones

Un sistema se puede modelar mediante la descripción de las restricciones o reglas (pre y poscondiciones) de las operaciones de los objetos que lo componen. Aquellas reglas se pueden abstraer a partir de los requerimientos o casos de uso del sistema. Existen cuatro pasos para escribir modelos de Pre-pos condición [1]:

- ✓ Escoger el objetivo fundamental para el desarrollo de la prueba.
- ✓ Diseñar y caracterizar las operaciones en el modelo.
- ✓ Diseñar las variables de estado en el modelo y elegir los tipos.
- ✓ Escribir las precondiciones y pos condiciones de cada operación.

Inicialmente se debe escoger un buen nivel de abstracción del modelo con el objetivo de obtener una buena caracterización de las entradas de la prueba y las salidas esperadas, al mismo tiempo se deben evitar detalles innecesarios que hagan el modelo más complejo de lo requerido. Luego se analizan cuáles elementos del SUT se controlarán y cuáles se observarán**,** y cómo esto puede ser mapeado en operaciones del modelo. Las relaciones entre los medios de observación y control del SUT**,** y las operaciones de observación y control del modelo de pruebas pueden ser [1]:

- ✓ Uno a uno, donde cada operación del modelo representa exactamente un medio de observación o control del SUT.
- ✓ Muchos a uno, donde cada medio de observación o control del SUT puede ser dividido en varias operaciones más simples en el modelo de pruebas, se pueden realizar diferentes operaciones dependiendo de los parámetros de entrada.
- ✓ Uno a muchos, donde cada operación del modelo de pruebas corresponden a muchos medios de control u observación de SUT, esto reduce la complejidad del modelo empaquetando varias operaciones del SUT en una sola operación del modelo de pruebas.
- ✓ Muchos a muchos combinando las anteriores, pero ésta casi no es usada debido a su complejidad.

Las precondiciones especifican las condiciones que deben ser verdaderas antes que las operaciones sean ejecutadas. A continuación un ejemplo con instrucciones en OCL (Object Constraint Language) [11, 20]:

Context Player*::calculateFinalScore():: Integer*

Pre: *self.isComplete = true*

La precondición relacionada a la operación "*calculateFinalScore*()" es que el jugador haya completado el juego. La poscondición especifica las condiciones que deben ser verdaderas después que las operaciones hayan sido ejecutadas. A continuación un ejemplo con poscondiciones [11, 20]:

Context GameEvent*:: processPlayerChoices():: Integer*

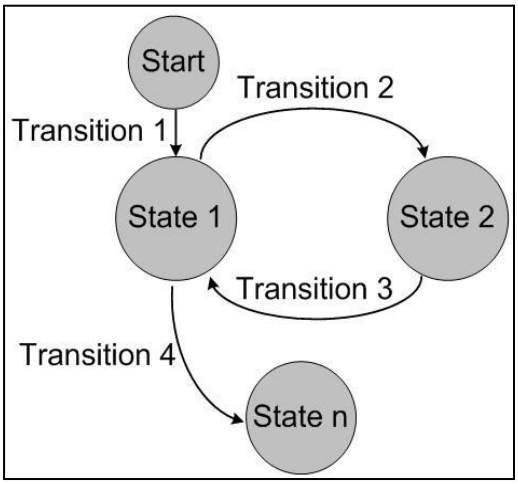
Pos*t: result = 0*

De acuerdo con este ejemplo, la poscondición para la operación "*processPlayerChoices*" es que el jugador no tenga escogencias para jugar.

2.2.1.2 Modelos Basados en Transiciones

El comportamiento del sistema se modela como las transiciones entre diversos estados debido a eventos. Usualmente es representado por una máquina finita de estados (FSM, por sus siglas en inglés) [1,19]. Gráficamente cada nodo de la FSM representa un estado en particular del SUT y cada arco representa una acción sobre el SUT (Ver Figura 1-5).

Figura 1-5: Modelado basado en transiciones y estados. [1,19].

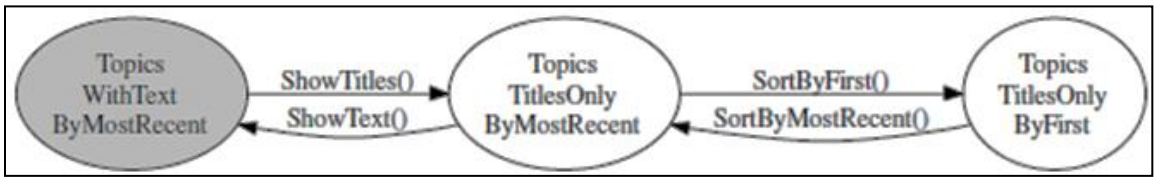


Una máquina de estados finita es una colección finita de transiciones de estados y puede estar conformada por escenarios. Un escenario es una secuencia ordenada y explicita de transiciones de estados lo cual define un camino y contiene estados iniciales y estados

de aceptación opcionales en el cual puede finalizar (si no se definen estados de aceptación los escenarios pueden finalizar en cualquier estado).

La Figura 1-6 visualiza un diagrama de transiciones de estados para un escenario en particular del sistema NewsReader [19], se comienza en el estado Topics WithText ByMostRecent y se aplica una acción ShowTiltes() lo cual conduce al estado Topics TitlesOnly ByMostRecent y luego se aplica una acción SortByFirst() lo que conduce al estado de aceptación final Topics TitlesOnly ByFirst. Si se comienza por este último aplicando las acciones SortByMostRecent() y ShowText() se regresará al estado inicial.

Figura 1-6: Escenario de una FSM para el sistema *NewsReader* [19].



Es importante aclarar que los efectos de una acción dependen de su estado actual. Una FSM representa el comportamiento compacto porque cada estado solo ocurre una vez, si el escenario o camino alcanza el mismo estado más de una vez, el diagrama contendrá ciclos (loops).

2.2.1.3 Modelos Basados en UML

Se emplean los diagramas UML (*Unified Modeling Language*) para modelar el comportamiento del sistema. El modelo de pruebas inicia con el diagrama de clases, el cual define una vista abstracta de la clase del SUT, los campos de datos que son utilizados en la clase y la relación con otras clases son útiles para la generación de pruebas, pero este diagrama no es suficiente para modelar el comportamiento dinámico del SUT.

Hay varios caminos para modelar el comportamiento completo del SUT con UML [1,17]: emplear el diagrama de máquinas de estado de UML para definir el comportamiento de objetos del SUT, el diagrama de actividades UML para definir operaciones complejas que se desarrollan alrededor de varias clases, usar precondiciones y post condiciones OCL para definir el comportamiento de los métodos declarados en los diagramas de clases.

A continuación se presenta un compilado de los diferentes diagramas de UML y su uso en el modelado para pruebas.

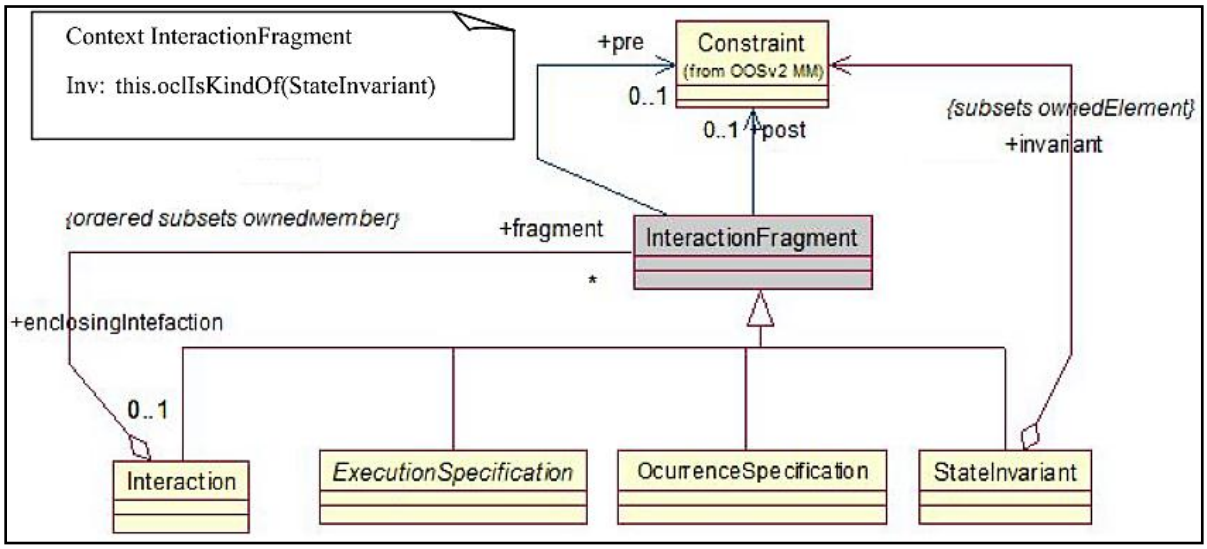
Tabla 1-4: Uso de UML para el Modelado de Pruebas [17].

Vista	Diagrama	Uso en el Modelado para Pruebas
Estática	Diagrama de Clases	Describe la estructura estática de un sistema, los datos dentro de cada clase y la asociación entre ellas. Se emplea para describir el modelo de pruebas.
	Diagrama de Objetos	Se usa para definir los estados iniciales del modelo para la generación de pruebas.
Requerimientos	Diagramas de Casos de Uso	Ofrece una visión general los requerimientos del sistema, pero no es suficiente para la generación de pruebas. Sin embargo cada especificación del caso de uso puede actuar como un objetivo de prueba informal de alto nivel, el cual guía el criterio de selección de pruebas.
Dinámica	Diagrama de Máquinas de Estados	Se emplea para modelar el comportamiento del sistema para propósitos de prueba. Cada transición describe un fragmento del comportamiento que se desea probar.
	Diagrama de Actividades	Se emplea para modelar procesos o flujos de trabajo. Los cuales son base para la generación de pruebas.
	Diagrama de Secuencias	Se usa para describir un caso de prueba abstracto, es decir indica las salidas que el sistema bebe tener. También es útil como notación para definir la selección de pruebas, pues puede especificar escenarios del comportamiento del modelo.
	Diagramas de Comunicación	Su uso es similar al diagrama de secuencias.

Un modelo bastante utilizado en este campo es el perfil de pruebas para UML (UML Testing Profile [17]), el cual define un lenguaje para diseñar, visualizar, especificar, analizar, construir y documentar los artefactos de un sistema de pruebas basándose en extensiones de UML. Este lenguaje se basa en el metamodelo de UML y reutiliza su sintaxis definiendo conceptos específicos para pruebas; dichos conceptos son agrupados en: comportamiento de las pruebas, arquitectura de las pruebas, datos de las pruebas y tiempo de la prueba [22].

En la Figura 1-7 se presenta un ejemplo de la extensión del metamodelo UML [22] para ser aplicado en pruebas. A la clase InteractionFragment se le agrega un Constraint de OCL y dos relaciones, una que representa la precondición del diagrama de secuencia y otra que representa su precondición. Se adiciona también una restricción que indica que dichas pre y poscondiciones no se aplican a InteractionFragment de tipo StateInvariant, ya que este representa invariantes donde las condiciones no cambian.

Figura 1-7: Extensión del UML para pruebas [22].



2.2.2 MBT aplicado a GUIs

Se mencionará una síntesis de los principales trabajos que se han realizado para automatizar las pruebas sobre interfaces gráficas de usuario.

S. Wieczorek *et al.* [8] presentan el estado del arte de la aplicación de las pruebas basadas en modelos para la automatización de pruebas de caja negra a interfaces de usuario y sistemas corporativos, con el ánimo de motivar la investigación en este campo específico. Argumentan que no existen muchos desarrollos respecto a un proceso robusto de pruebas automáticas sobre interfaces de usuario debido principalmente a que es complejo modelar el gran número de estados posibles que pueden llegar a tener las interfaces gráficas de usuario modernas. Mencionan que los avances más completos respecto al tema es la generación de casos de prueba a interfaces gráficas de usuario empleando un modelo de flujo de eventos o diagramas UML anotados. Por último afirman

que se debe centrar un gran esfuerzo en esta área para garantizar la calidad y confiabilidad en aplicaciones corporativas orientadas a servicios.

D. Sevilla *et al.* [3] diseñaron e implementaron una herramienta para generar pruebas automáticas de software sobre interfaces gráficas de usuario (denominada Open-HMI Tester), la arquitectura de esta aplicación es flexible, lo cual significa que es independiente del sistema operativo o tecnología de ventanas que se puede llegar a emplear en el software bajo prueba. Su diseño se basa en unos módulos que implementan la funcionalidad general (encargados de controlar el proceso de captura y reproducción de casos de prueba) y otros que deben ser adaptados con el fin de concretizar la funcionalidad específica del sistema (encargados de la comunicación directa con la interfaz gráfica de usuario). La principal ventaja de la herramienta es que permite acceder al núcleo de la aplicación a probar, pues soporta la introspección no intrusiva del código, y por lo cual es capaz de acceder a la información de cualquier evento de la interfaz gráfica, con esto se logra reproducir una secuencia de eventos que realmente simulan la interacción del usuario con el componente, soportando modificaciones moderadas de la interfaz (cambio de localización, intrusión de nuevos elementos, etc.). Como trabajo futuro los investigadores pretenden plantear una arquitectura más general que permita extender las pruebas de la interfaz de usuario hacia la lógica de negocio y con la evaluación de la usabilidad de la aplicación.

D. Sevilla *et al.* [7] proponen un método para automatizar la generación de pruebas en interfaces gráficas de usuario basado en el desarrollo de casos de prueba anotados, este enfoque es un método intermedio entre las pruebas de software basadas en modelos y las que no emplean ningún modelado. Esta propuesta se apoya en un proceso de autogeneración de casos de prueba, para lo cual emplea los siguientes elementos: - un conjunto de casos de uso que describe el comportamiento de la interfaz, - un conjunto de anotaciones que describen las variaciones que pueden llegar a afectar los diferentes elementos que componen la interfaz, y – una serie de reglas de validación para comprobar los valores de las propiedades de ciertos elementos. El proceso descrito tiene como punto de partida los casos de uso (secuencias de eventos realizadas sobre la GUI), del cual se auto-genera un conjunto de casos de prueba, teniendo en cuenta los puntos de variación y las reglas de validación incluidas en las anotaciones, que se ejecutan sobre la interfaz gráfica de usuario. Esta solución evita el proceso manual (demorado y

costoso) de crear y mantener un modelo completo de la GUI, ya que solo prueba los aspectos fundamentales de ésta mediante las anotaciones de los elementos más representativos, con esto se logra un proceso de pruebas y de desarrollo más ágil. Como trabajo futuro se pretende extender el proceso de pruebas de la GUI a la lógica de la aplicación y ejecutar el conjunto de pruebas de forma paralela en un ambiente distribuido.

Yongzhong Lu *et al.* [11] presentan un método para automatizar las pruebas sobre interfaces gráficas de usuario basado en el diagrama de flujo de eventos de la misma. Para este método se desarrolló una herramienta la cual realiza ingeniería inversa sobre un prototipo o ejemplo de prueba de la GUI, con el objeto de extraer un diagrama de flujo de eventos que describa su comportamiento potencial. Luego, basado en el diagrama anterior, se desarrolló e implementó un algoritmo optimizado, aplicando aprendizaje de máquina y procesos estocásticos, para hallar los mejores casos de prueba que se ejecutarán sobre la nueva interfaz. Con este proceso inteligente y automático se disminuye considerablemente el esfuerzo manual, ya que solo se aplica supervisión asistida en las fases: de refinación del modelo generado (el diagrama de flujo de eventos), de generación de los oráculos de prueba y de análisis de resultados (por qué fallaron los casos de prueba); lo que permite una gran flexibilidad y adaptabilidad a variaciones del software y gestión sistemática de la cobertura de los casos de prueba. Para probar el método se desarrolló un framework para ayudar a la automatización del proceso.

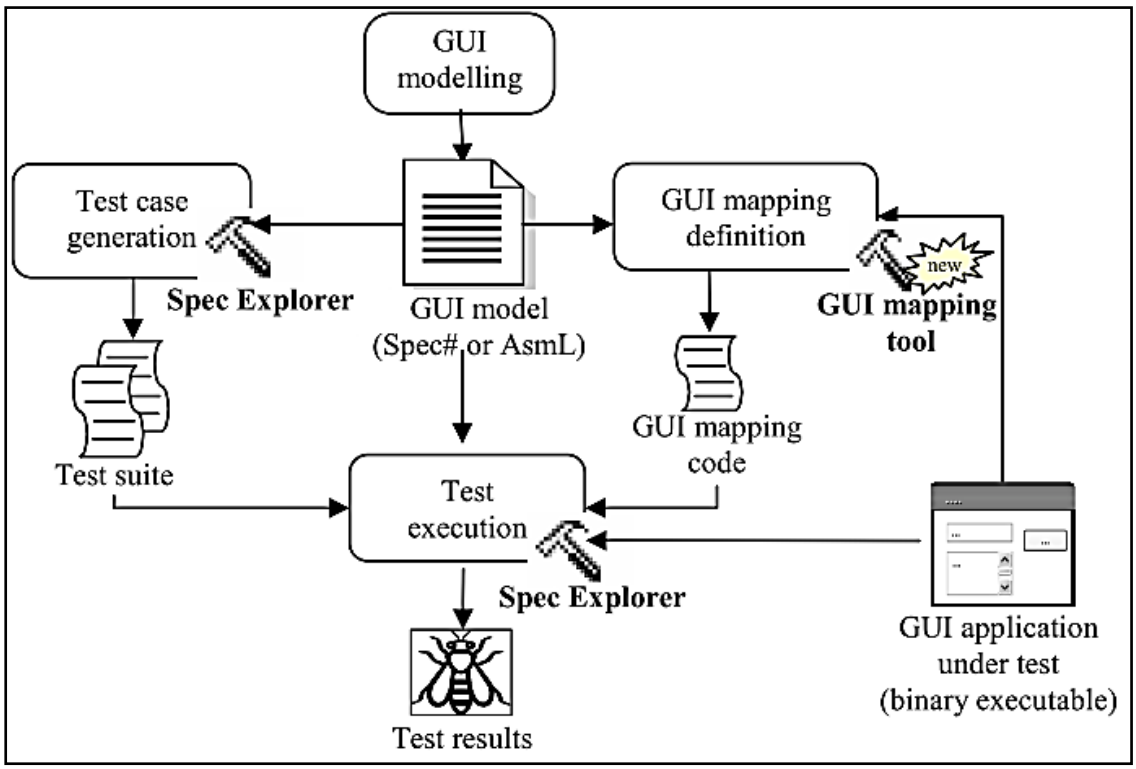
Ana Cristina Paiva [12], [14], [24], presenta una forma para automatizar las pruebas sobre las interfaces gráficas de usuario de las aplicaciones, empleando una extensión de la herramienta *Spec Explorer* y el lenguaje *Spec#* para realizar la especificación formal. La herramienta *Spec Explorer* (Anexo A Anexo: Spec Explorer, [15]), desarrollado por Microsoft Research, soporta la generación y ejecución automática de casos de prueba, pero requiere que las acciones descritas en el modelo sean enlazadas con los métodos de la implementación.

El método está diseñado para interfaces graficas de usuario de estructura jerárquica y busca probar que las respuestas de la interfaz cuando el usuario interactúa con ésta sean las esperadas por los casos de prueba. Los casos de pruebas se enfocan en que las ventanas invocadas, las emergentes de alerta y las de confirmación sean las adecuadas cuando el usuario realiza alguna acción. Con el método propuesto también se

pueden probar interfaces ya desarrolladas, puesto que se realiza un paso de ingeniería inversa en donde se automatiza la forma de obtener la información estructural de objetos que componen la GUI y que se acoplan con las acciones de usuario descritas en el modelo; con base en esta información se genera un intermediario que simula las acciones de la GUI de la aplicación bajo prueba.

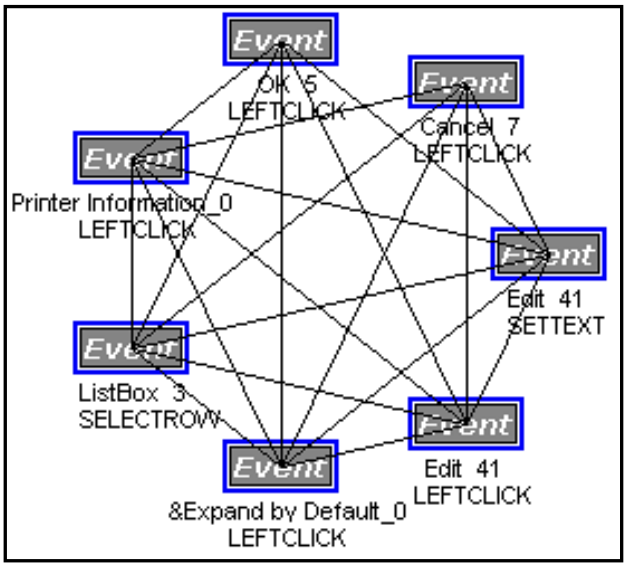
En la Figura 2-8 se presenta las actividades y artefactos principales involucrados en el proceso de pruebas para GUIs con Spec Explorer utilizando una herramienta de mapeo que enlaza las acciones del modelo con los métodos de la implementación de manera ágil.

Figura 2-8: Proceso de Modelado y ejecución de Pruebas de la GUI empleando el intérprete GUI Mapping Tool [12, 22, 24].



En [9, 10] se propone modelar la GUI a partir de su gráfica de flujo de eventos (*Event-Flow Graph*). Una Gráfica de Flujo de Eventos representa todas las posibles interacciones entre los eventos de una GUI. Este diagrama contiene todos los eventos que podrían ser accedidos en un punto determinado del tiempo, con lo cual se pueden especificar escenarios, y basándose en esto generar casos de prueba, ejecutar los casos de prueba, verificar efectividad de las pruebas y obtener reportes de cobertura. En la Figura 2-9 se muestra una gráfica de flujo de eventos para el escenario de conexión a impresora de la interfaz de WordPad, un link de un evento a otro significa que el segundo puede ser ejecutado luego del primer evento.

Figura 2-9: Gráfica de Flujo de Eventos para el escenario de conexión a impresora de WordPad [10].



3. Implementación de MBT para GUIs

3.1 Estrategia Desarrollada

En la primera etapa del trabajo se definió la herramienta que se iba a utilizar para dar soporte a todo el ciclo del proceso de pruebas basadas en modelos. La selección de la herramienta es expuesta en la sección 3.3.1. Una vez especificada la herramienta base se definió la estrategia que se emplearía para la implementación de MBT sobre GUIs.

Para poder alcanzar los objetivos propuestos se empleó una estrategia iterativa. Se realizaron tres iteraciones y en cada una de ellas se implementó un caso de ejemplo para una vista de usuario particular (Anexos B, C, D), aplicando un ciclo completo de las pruebas basadas en modelos. En cada iteración se iba aumentando la complejidad de la vista y el número de funcionalidades que está expone al usuario.

Cada iteración estuvo compuesta por las siguientes fases:

- ✓ En la primera se definió un conjunto básico de requerimientos iniciales, a partir de los cuales se concretó los aspectos que se modelarían de la interfaz.
- ✓ En la segunda, con base en los aspectos definidos y apoyándose en la herramienta seleccionada (ver sección 3.3.1), se generó un modelo explícito de pruebas que modelase, desde alguna perspectiva, el comportamiento de la vista de usuario definida.
- ✓ En la tercera etapa, una vez implementado el modelo, con la herramienta se generaron de manera automatizada las pruebas abstractas o de alto nivel. Éstos son una representación lógica de los casos de prueba definidos.
- ✓ En la cuarta, se concretizaron los casos de prueba. Empleando la herramienta se generó de manera automatizada la suite de pruebas ejecutable.
- ✓ En la quinta, se desarrolló el prototipo concreto de la interfaz gráfica. Utilizando la herramienta se ejecutaba de manera automática la suite de pruebas generada

sobre la implementación, y gradualmente se iba desarrollando el prototipo de la GUI hasta que se pasaran todas las pruebas. A medida que se iba refinando el modelo en cada iteración, también se iba refinando la implementación.

- ✓ En la quinta se efectuó una ejecución manual de pruebas (validación de usuario final), con el fin de comparar los resultados obtenidos en el proceso de generación y ejecución automática de pruebas. Si en esta fase se descubrían hallazgos o errores, que no se detectaban en el proceso automático, se volvía a ejecutar una nueva iteración, refinando las fases más afectadas.
- ✓ Finalmente se obtenían los resultados respecto a cobertura y efectividad de la iteración.

3.1.1 Selección Herramienta

En [26] y [31] se presenta un compilado detallado de herramientas comerciales y libres existentes para implementar el proceso de pruebas basadas en modelos en un proceso de desarrollo de software. Específicamente en [26] se realizó una comparación detallada de las capacidades de las principales herramientas, en términos de cobertura de pruebas, nivel de automatización y generación de pruebas.

En la Tabla 3-1 se despliega un breve consolidado de las principales herramientas empleadas en MBT ([26] , [31]). Para cada una se describe su categoría (si es un proyecto interno, open source, está a cargo de un departamento de investigación o si es comercial), el tipo del modelo a partir del cual se generan los casos de prueba (basado en una máquina de estados finita o en UML) y la tecnología o plataforma objetivo que maneja.

Tabla 3-1: Descripción de las principales herramientas en MBT [31].

HERRAMIENTA	CATEGORIA	TIPO DEL MODELO	PLATAFORMA OBJETIVO
GOTCHA-TCBeans	Interna de IBM	FSM	C/C++, Java
MBT	Open source	FSM / EFSM (Extended FSM)	General
MOTES	Research	EFSM	General
TestOptimal	Comercial	FSM	General
AGEDIS	Research	AML (Basado en UML)	C/C++, Java
ParTeG	Research	UML	Java
Qtronic	Comercial	UML	General
Test Designer	Comercial	UML	General
Spec Explorer 2010	Research	FSM	General

Como el desarrollo del trabajo debe servir al crecimiento profesional, los criterios principales para la selección de la herramienta fueron:

- ✓ Permitir cubrir todo el ciclo de MBT. Desde la construcción del modelo hasta la generación de código ejecutable de pruebas.
- ✓ Modelar el comportamiento del SUT por medio de algunas de las técnicas existentes.
- ✓ Poseer integración con *Visual Studio Ultimate 2010*, ya que a nivel profesional se tiene experiencia en el manejo de este IDE de la plataforma *.Net* de Microsoft.

La herramienta que cubre todos los criterios anteriores es *Spec Explorer 2010* de Microsoft; es por ello que se seleccionó como la herramienta base para desarrollar los tres casos de ejemplo propuestos.

3.2 Diseño del Modelo de Pruebas de la GUI

Para efectos del trabajo las GUIs serán denominadas vistas de usuario, las cuales forman parte de la capa presentación de una aplicación. Durante esta sección y las posteriores de este capítulo se expondrá, por medio de un ejemplo, la propuesta finalmente definida para construir el modelo de pruebas de una vista de usuario en particular a partir del cual se generen los casos de prueba. La propuesta planteada se presentará de manera generalizada para vistas simples y compuestas, y se concretizará para el ejemplo.

La interfaz gráfica de usuario que se utilizará como ejemplo para aplicar el proceso de pruebas basadas en modelos es la vista simple mostrada en la Figura 3-1**.** Esta vista es un formulario para el ingreso de los siguientes datos de una empresa:

- ✓ *Nit*: Número que identifica a la empresa.
- ✓ *Nombre:* Texto que representa el nombre de la empresa.
- ✓ *Representante Legal*: Texto que representa el representante legal de la empresa.
- ✓ *Teléfono*: Texto que representa el número telefónico de la empresa.
- ✓ *Dirección*: Texto que representa la dirección de la empresa.
- ✓ *Fecha de Vinculación:* Fecha de afiliación a una caja de compensación familiar

Figura 3-1: Interfaz Gráfica de Usuario para el ingreso de datos de una empresa. Denominada Formulario de Empresa.

Para diseñar el modelo de pruebas de la GUI, lo primero que se definieron son los requerimientos o validaciones que debe implementar la interfaz. A partir de esto se abstraen los aspectos que se moldearán de la interfaz, de modo que se comporte de acuerdo a los requerimientos.

3.2.1 Definición de los Requerimientos Base

Después de realizar los casos de ejemplo de implementación del proceso MBT, cuya descripción detallada se puede consultar en los Anexos B, C y D, se obtuvo el consolidado de los requerimientos y funcionalidades básicas que se probaron de las interfaces gráficas. La definición de los requerimientos generales se definió en términos de las validaciones que se deben probar de la GUI. Las validaciones que se tuvieron en cuenta a la hora de probar una interfaz gráfica de usuario son:

- ✓ ***Validación de campos obligatorios*:** Se prueba que los campos de texto obligatorios de entrada no se encuentren vacíos antes de que el usuario ejecute una acción en la espera de un resultado, en donde se valida que exista algún mensaje de advertencia al usuario indicando el error en el campo. Por ejemplo el usuario desea realizar una suma pero no digita ningún dato en los campos de entrada de los sumandos y pulsa el botón de resultado, entonces la respuesta de la interfaz es desplegar un mensaje indicando que los campos no deben estar vacíos.
- ✓ ***Validación de la longitud/rango apropiada de los campos:*** Se prueba que los campos de texto de entrada tengan la longitud apropiada según los requerimientos iniciales antes de que el usuario ejecute una acción en la espera de un resultado, en donde se valida que exista algún mensaje de advertencia al usuario indicando el error en el campo. Si el campo es numérico se valida que se encuentre dentro del rango definido. Por ejemplo el usuario digita el Teléfono de una empresa y se valida que éste no tenga más de 15 caracteres, si no cumple la condición entonces la respuesta de la interfaz es desplegar un mensaje indicando que el campo no tiene la longitud apropiada.
- ✓ ***Validación del formato de los campos de entrada de la interfaz:*** Se prueba que los campos de texto de entrada tengan el formato apropiado según los requerimientos iniciales antes de que el usuario ejecute una acción en la espera

de un resultado, en donde se valida que exista algún mensaje de advertencia al usuario indicando el error en el campo. Por ejemplo el usuario digita el Nit de una empresa y se valida que éste sea numérico, si no cumple la condición entonces la respuesta de la interfaz es desplegar un mensaje indicando que el campo no tiene el formato apropiado.

- ✓ ***Validación personalizada de datos:*** Se prueba las reglas de validación especificadas para algunos campos. Estas reglas son aquellas que puedan ser manejadas desde la lógica de presentación, por ejemplo validar que la Fecha de Vinculación ingresada sea menor a la fecha actual de sistema.
- ✓ ***Validación del despliegue de ventanas, diálogos y mensajes:*** En este caso el objetivo es probar que las ventanas invocadas, las emergentes de alerta y las de confirmación sean las adecuadas cuando el usuario realiza alguna acción (enfoque propuesto en [24]). Es decir verificar en todo momento luego de la interacción del usuario cuáles ventanas deben estar activas. Por ejemplo probar que efectivamente luego de pulsar el botón de Buscar del Menú de Edición efectivamente se despliegue el dialogo de Búsqueda o si se está realizando una operación de guardar un archivo y éste ya existe, se verifique que se despliegue un MesaggeBox al usuario indicando si desea o no sobrescribir el archivo. Con esto se prueba que las invocaciones de una interfaz compuesta sean las apropiadas.
- ✓ ***Validación de la apropiada activación de controles de la interfaz*****:** Probar que un control determinado pueda o no responder a la interacción del usuario, ya que según el estado de la interfaz a veces es necesario deshabilitar un control para restringir su uso. Por ejemplo deshabilitar el botón de guardar para evitar que el usuario haga clic sobre él si existe algún error en la validación de campos de un formulario.
- ✓ ***Validación de la interacción y comunicación apropiada entre los diferentes componentes de la interfaz*****:** Probar que cuando un control o componente se comunique o interactúe con otro haya efectivamente alguna respuesta o acción por parte del segundo control. Por ejemplo cuando se adicione una empresa desde un formulario de edición de datos, se actualice la lista de empresas del control principal.
- ✓ ***Validación de invocación de operaciones ante una acción específica de usuario:*** Como solo se pretende probar la lógica de presentación de la GUI,

basta con validar que efectivamente la interfaz responda ante eventos disparados por el usuario. Sin embargo en algunas ocasiones se espera que la aplicación ejecute alguna acción manejada por la lógica del negocio o la persistencia de datos, por ejemplo en la acción de guardar un elemento modificado se espera que elemento finalmente se guarde en disco. En estos casos no se prueba que se haya modificado la fuente datos, sino que por lo menos se invoque la operación(es) de alto nivel que desencadenarían la ejecución de esta acción, ya que esto no es responsabilidad de la capa de presentación sino de las capas inferiores.

- ✓ ***Validación de alto nivel de los escenarios principales de la interfaz:*** Para ejecutar varias de las validaciones o pruebas descritas anteriormente en un solo proceso se realizan escenarios que emulen sobre la interfaz diferentes acciones secuenciales de usuario. Por ejemplo emular sobre una interfaz de lista de contactos las acciones de seleccionar un ítem, luego abrir el dialogo de búsqueda, ingresar cierto tipo de entradas, realizar una búsqueda y luego cancelar.

Para el caso de la Figura 3-1**,** las validaciones definidas para los principales elementos gráficos o controles que componen esta interfaz son mostradas en la Tabla 3-2. El tipo de validación indica que pertenece a algún grupo descrito anteriormente.

Tabla 3-2: Validaciones definidas para el Formulario de Empresa.

Elemento	Tipo de Validación	Descripción
Campo Nit	Validación de campo obligatorio.	El Nit no puede ser vacío.
	Validación de formato.	El Nit debe ser numérico.
	Validación de rango.	El Nit debe ser >= 0 y <= 99999999999999
Campo Nombre	Validación de campo obligatorio.	El Nombre no puede ser vacío.
	Validación de longitud.	El Nombre debe tener una longitud <= 20 caracteres
Campo Representante Legal	Validación de campo obligatorio.	El Representante Legal no puede ser vacío.
	Validación de longitud.	El Representante Legal debe tener una longitud <= 20 caracteres
Campo Dirección	Validación de campo obligatorio.	La Dirección no puede ser vacía.
	Validación de longitud.	La Dirección debe tener una longitud <= 20 caracteres
Campo Teléfono	Validación de campo obligatorio.	El Teléfono no puede ser vacío.
	Validación de longitud.	El Teléfono debe tener una longitud <= 15 caracteres
Campo Fecha de Vinculación	Validación de campo obligatorio.	La Fecha no puede ser vacía.
	Validación personalizada.	La Fecha debe ser menor o igual a la fecha actual.
Botón Guardar	Validación de activación.	Si no existen errores de validación se activa el botón guardar.
	Validación de despliegue.	Validar que luego de oprimir el botón de guardar se despliegue un cuadro de mensaje indicando al usuario que el proceso fue realizado satisfactoriamente.
Formulario de Empresa	Validación de despliegue.	Validar si el formulario está activo para interactuar con el usuario.
	Validación de invocación.	Validar la invocación la operación de guardar.
	Validación de escenario.	Probar un escenario completo de edición de datos de una compañía con diferentes valores. El escenario es: Abrir el formulario, ingresar los datos de la empresa, guardar la información, esperar la respuesta del formulario y cerrar el formulario.

3.2.2 Definición de los Aspectos del Comportamiento a Modelar

Para poder cumplir con los requerimientos o validaciones establecidas en la sección anterior es necesario definir los aspectos que se modelarán de una interfaz gráfica de usuario. Al concretar estos aspectos se puede diseñar el modelo o modelos necesarios para aplicar el proceso MBT.

El número de estados internos de una interfaz gráfica de usuario y rutas posibles de interacción del usuario puede llegar a ser muy grande dependiendo de la composición de la interfaz, ya que en un punto determinado de ejecución el estado de la interfaz es definido por los estados o valores de las propiedades de todos los componentes, controles y sub elementos que la conforman [10].

El enfoque propuesto para disminuir esta complejidad es abstraer dichos valores en estados más generales, es decir especificar propiedades en términos de la lógica de la capa de presentación y que estén definidas por un grupo determinado de características específicas de varios elementos. El comportamiento de dichas variables compondrá el estado de la interfaz y son las que se observarán durante la ejecución de la GUI, pues son descritas con base en los requerimientos. Por ejemplo se puede definir una propiedad que indique si todos los valores de entrada de los campos de un formulario son válidos para poder realizar una acción de actualización, es decir este atributo indica si es válida o no la interfaz para realizar cierto proceso. En el modelo se puede especificar una propiedad general que represente el consolidado de las variables que componen el estado de la interfaz.

Otro aspecto que es necesario definir son las acciones que se modelarán, es decir las acciones que compondrán el modelo. Estas acciones son modeladas con base en las funcionalidades que expone la interfaz.

Con base en el análisis de los requerimientos de los casos de ejemplo (Anexos B, C y D) y de las validaciones definidas, los aspectos que se modelaron de una interfaz gráfica de usuario son:

- ✓ **Propiedades que modelan el estado de la interfaz:** Estas propiedades indican si la interfaz ha alcanzado un estado definido por la validación de ciertas condiciones de las propiedades de los controles que la componen. Por ejemplo se modelan

propiedades como *IsValid* o *IsDirty* (Anexos B y D); la primera indica si los valores de entrada de un conjunto de campos son válidos y la segunda indica si ha habido modificaciones en los valores originales de los campos desde que se desplegó la interfaz. También se puede modelar un estado que indique cierta característica específica de la interfaz, como por ejemplo el Item actual seleccionado, un color de representación, el tipo de resultado de un proceso, etc.

- ✓ **Propiedades que modelan la habilitación de controles:** Estas propiedades indican si cierto conjunto de controles deben estar habilitados o no para que el usuario pueda interactuar con ellos. Por ejemplo la propiedad *HabilitarGuardar* (Anexo D) activa el botón de guardar siempre y cuando todos los campos de entrada cumplen con las condiciones de validación definidas o ha habido alguna modificación en los datos.
- ✓ **Propiedad que modela las ventanas activas**: Esta propiedad indica de manera jerárquica las ventanas (formularios, diálogos o cuadros de mensajes) que están activas en punto determinado de ejecución. Por ejemplo si se abre un diálogo de búsqueda y luego se realiza una operación de búsqueda y se despliega un cuadro de mensaje indicando que no hubo resultados, entonces con esta propiedad se sabrá que luego de esta acción debe activarse un cuadro de mensaje de información al usuario (Anexos C y D).
- ✓ **Propiedad que modela los errores de validación:** Esta propiedad indica los campos que poseen errores de validación. Los mensajes de errores de validación deben ser desplegados al usuario. Por ejemplo la información de validación de datos de entrada de los campos de un formulario puede desplegarse dinámicamente por un *errorProvider* o por un cuadro de mensaje de advertencia (Anexo D).
- ✓ **Acciones que modelan el manejo de la interfaz**: Se verifica que se invoquen las acciones comunes del manejo de la interfaz; como por ejemplo, abrir, aceptar, cancelar o mostrar otras ventanas, controles o diálogos (Anexos C y D).
- ✓ **Acciones que modelan los escenarios base de las funcionalidades:** Modela como tal la funcionalidad objetivo de la interfaz, por ejemplo para el dialogo de Búsqueda se modela una acción que represente el escenario de búsqueda de esta interfaz (Anexos C y D).

Para el caso de la Figura 3-1**,** los aspectos que se modelaron para el Formulario de Empresa son descritas en la Tabla 3-3. Estos aspectos son mapeados a propiedades o

acciones concretas en el modelo a partir de las cuales se define el comportamiento deseado de la interfaz. El tipo de aspecto a modelar indica que pertenece a un grupo descrito anteriormente.

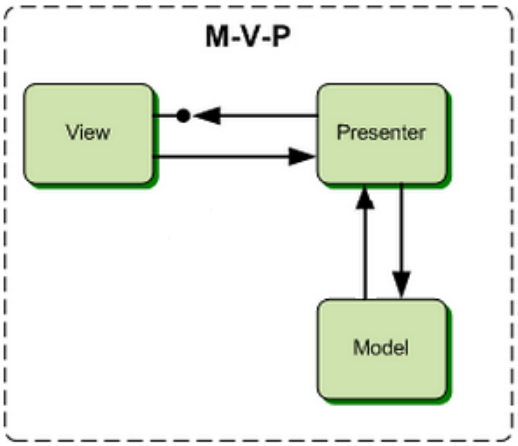
Tabla 3-3: Aspectos modelados para el Formulario de Empresa.

Aspecto a Modelar	Tipo	Representación en el Modelo	Descripción
Estado de la interfaz	Propiedad	IsValid	Indica si el formulario tiene un estado válido o no. Es válido si los datos de entrada de la empresa cumplen con las reglas de validación definidas.
		IsDirty	Indica si ha habido modificaciones en los valores originales de los campos desde que se desplegó el formulario.
		CampoConErrores	Indica los campos que poseen errores de validación.
		Items	Indica el número de empresas existentes. Al ingresar una empresa este valor debe aumentar en uno.
Habilitación de controles	Propiedad	HabilitarGuardar	Indica si debe estar o no habilitado el botón de guardar. Ésta habilitado si los datos de entrada son correctos y ha habido modificaciones, es decir es una combinación de IsValid e IsDirty.
Ventana activa	Propiedad	ActiveWindow	Indica el nombre de la ventana activa que puede interactuar con el usuario.
Manejo de la interfaz	Acción	LaunchView	Acción para abrir la interfaz principal que contiene al formulario de empresa.
		Close	Acción para cerrar la interfaz principal que contiene al formulario de empresa.
		AddNewCompany	Acción de abrir el Formulario de Empresa
		Cancel	Acción de cerrar el Formulario de Empresa
		CompanyViewScenario	Acción de ingresar datos al formulario de empresa.
Funcionalidad Base	Acción	Guardar	Acción de guardar los datos de la empresa.
		MsgAckProceesOk	Acción de informar al usuario que el proceso de guardar los datos de la empresa fue satisfactorio.

3.2.3 Modelo Concreto de Pruebas

El enfoque está dirigido a probar interfaces graficas de usuario que tengan separada la lógica de presentación de la lógica de negocio. Uno de los patrones más utilizados para diseñar la GUI con este concepto, es el patrón MVP (*Model View Presenter* - Figura 3-2) [29], pues brinda un desacople entre la tecnología de interfaz y la lógica de la aplicación, demás que es muy utilizado en los procesos de desarrollo de software orientado a pruebas [25, 30].

Figura 3-2: Estructura del Patrón MVP (Model View Presenter) [25].



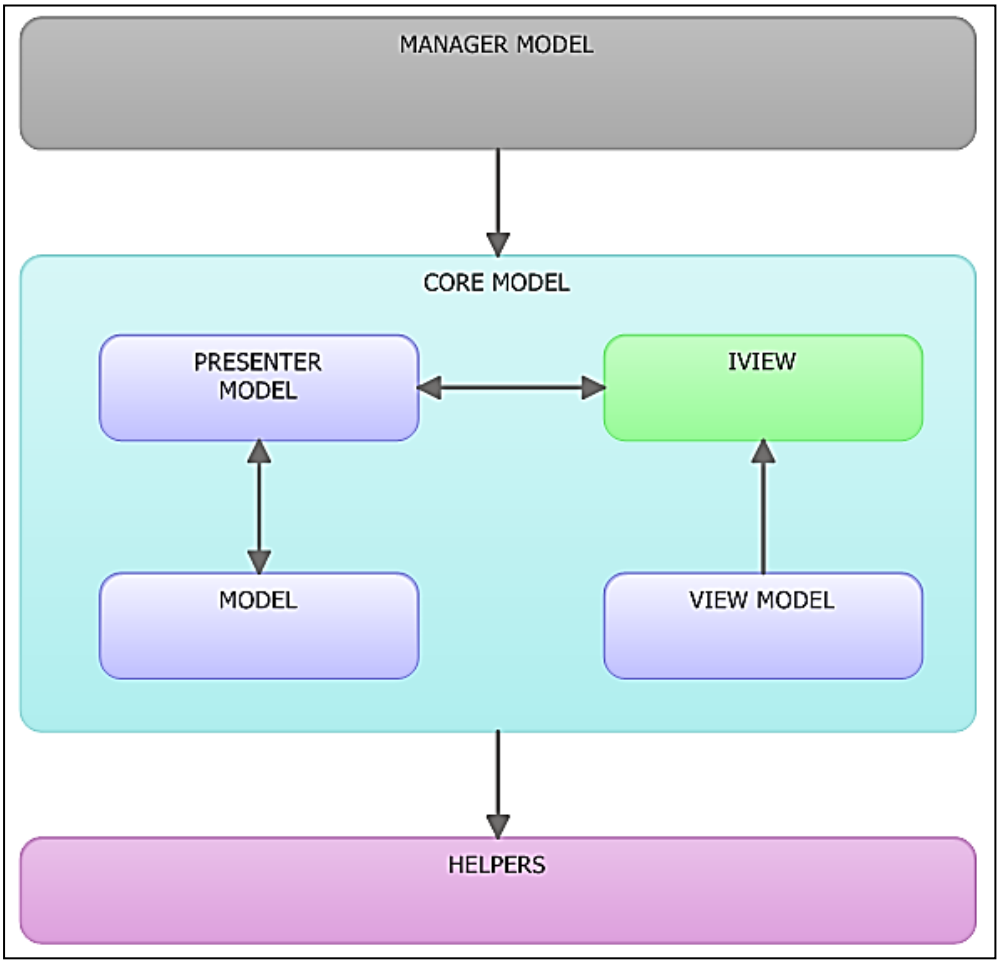
En la Figura 3-2 se presenta la estructura general del patrón MVP [25, 29, 30], en donde el modelo es la lógica de negocio, la vista es la interfaz gráfica de usuario y el presentador es el encargado de desacoplar la comunicación entre el modelo y la vista. El desacople es tal, que el presentador solo podrá interactuar con la vista por medio de una interfaz, es decir el presentador no conocerá los detalles de implementación de la vista de usuario. El presentador es el único encargado de actualizar y manipular la vista, y es allí donde radica la lógica de presentación, pues la vista es pasiva y solo se encarga de pasar los eventos disparados cuando el usuario interactúa con esta. Las características más destacables de MVP son: mantenibilidad del código, desacoplamiento y facilidad de implementación de pruebas de software [25, 29, 30].

Por las características que brinda el patrón MVP, se seleccionó para implementar MBT sobre GUIs, en donde se enfocará en probar el comportamiento adecuado del presentador que conlleva, implícitamente, probar el comportamiento final de la GUI.

Ahora bien, teniendo claro el enfoque, los requerimientos mínimos (sección 3.2.1) y aspectos a modelar (sección 3.2.2), se concreta la manera de modelar la interfaz gráfica de usuario para poder aplicar el proceso MBT, teniendo en cuenta las características que nos ofrece la herramienta escogida para la implementación (*Spec Explorer 2010- Anexo A*); como por ejemplo poder definir un modelo por medio de un lenguaje específico y explorar su comportamiento a través de una máquina finita de estados.

Luego de refinar el modelado de forma iterativa al implementar los casos de ejemplo (Anexos B, C y D) y de tener en cuenta el planteamiento anterior, finalmente la arquitectura base propuesta para generar el modelo concreto del proceso de pruebas basadas en modelos sobre interfaces graficas de usuario es mostrado en la Figura 3-3.

Figura 3-3: Arquitectura base propuesta para el modelo concreto de pruebas del proceso MBT sobre GUIs.



En la estructura presentada en la Figura 3-3**,** el *Manager Model,* el *Presenter Model*, el *View Model*, el *Model* y el *IView* son clases u objetos específicos del modelo, mientras que *Helpers* es una capa compuesta por varios objetos de propósito general. Con este esquema se busca probar el comportamiento de la vista (GUI) a través del presentador, pues éste tiene como responsabilidad gestionar el comportamiento de la vista y es allí donde se validan y verifican los aspectos modelables que se han definido.

En la fase de modelado el *Presenter Model* y el *View Model*, e son los objetos principales que establecen el comportamiento que debería tener la implementación (SUT) y es a partir de esta especificación que se generan finalmente los casos de prueba. A continuación se describe con más detalle los elementos que participan en la estructura del modelo definido en la Figura 3-3.

- ✓ ***Manager Model:*** Es una clase que tiene como responsabilidad gestionar el acceso al resto de funcionalidades del modelo; primordialmente maneja el acceso al *Presenter Model*. Su papel es similar al de una fachada, pues proporciona una interfaz unificada de alto nivel para un conjunto de clases, en este caso el *Core Model*. Allí se especifican pre y pos condiciones para cada operación definida.
- ✓ ***Presenter Model:*** Es el componente principal del modelado ya que gestiona el comportamiento de la vista y manipula el modelo (datos). Es donde está implementa la lógica de presentación y básicamente detalla pos condiciones para cada operación definida. La fase de modelado para GUIs se enfoca en probar el comportamiento adecuado del PresenterModel, pues a través de éste se prueba el comportamiento de la GUI al acceder a las operaciones expuestas por la interfaz IView.
- ✓ ***IView:*** Esta interface expone las funcionalidades y atributos que debe tener la vista o interfaz gráfica de usuario concreta, de acuerdo a la abstracción de los requerimientos, y que serán empleadas por del *Presenter Model*.
- ✓ ***View Model:*** Este objeto implementa las funcionalidades del IView. Es la representación de la vista en la fase de modelado y pruebas, por lo cual este objeto simula el comportamiento real de la GUI, al emular las acciones realizadas por el usuario; sin embargo estas acciones son activadas por el *PresenterModel*.
- ✓ ***Model:*** Este objeto contiene los datos y la lógica de negocio. Con el enfoque propuesto no es de interés probar esta capa, así que en los casos de ejemplo

propuestos esta se empleó como una capa de DTOs (*Data Trasnfers Objets*), en el sentido que no implementan ninguna lógica. Si fuera necesario, por objetivos de prueba de la interfaz, acceder a esta lógica, el *PresenterModel* emularía este comportamiento. Este elemento es empleado en fase de modelado como en fase de implementación.

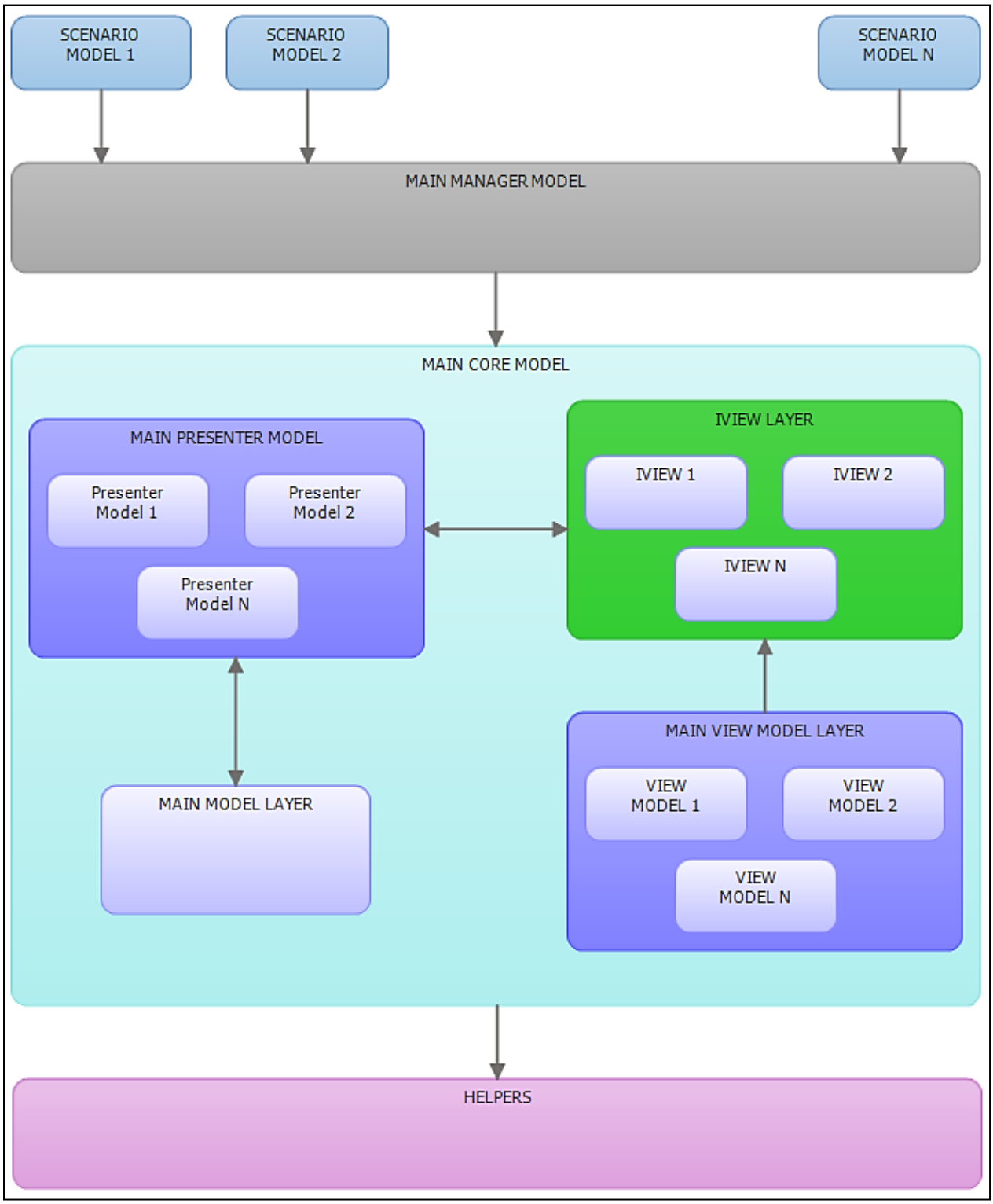
- ✓ ***Helpers:*** Esta capa expone utilidades que dan soporte a las capas superiores, es decir contiene clases de uso genérico que pueden ser empleadas por cualquier otro objeto. En algunos casos gestiona funcionalidades genéricas en alto nivel que pueden ser útiles para dar soporte a las pruebas de GUIs, como por ejemplo la administración de ventanas o archivos. En los casos de ejemplo implementados (Anexos C y D), los helpers principales son el *WindowManager* y el *FileManager*.

Para interfaces gráficas de usuario compuestas el número de estados puede llegar a ser muy grande, de acuerdo con los eventos disparados por el usuario, entonces no es una buena estrategia construir un solo modelo que represente toda la interfaz, ya que sería muy complejo de manejar, interpretar y mantener. Es por ello que la estrategia recomendada en [24] es realizar un diseño compuesto por diferentes vistas o perspectivas del modelo general que están relacionadas a ciertos escenarios. Los escenarios del modelo son obtenidos a partir de los requerimientos y casos de uso y se propone para esta clase de vistas compuestas la estructura mostrada en la Figura 3-4**.**

El planteamiento de la Figura 3-4 es más general y encapsula en alto nivel la estructura base propuesta (Figura 3-3). Las diferencias primordiales son:

- ✓ Se Implementa otra capa superior de modelado que contiene los objetos que representan los escenarios principales de la interfaz (*Scenarios Models*).
- ✓ El *Main Presenter Model* es, además de ser un *Presenter Model* de la vista principal, un administrador de todos los *Presenters Models* que manejan las vistas hijas que componen la interfaz principal.
- ✓ Las interfaces de las vistas, las vistas y los modelos se agrupan en capas.

Figura 3-4: Arquitectura base del modelo concreto de pruebas para vistas compuestas.



Con Spec Explorer 2010 el modelo se construye a partir de reglas (Anexo A), las cuales representan las acciones a ser modeladas. En la estructura descrita en la Figura 3-4 las reglas son definidas en los ScenarioModels.

Para el ejemplo definido al inicio de la sección, se especifica un escenario que consiste en utilizar el formulario de empresa para adicionar una nueva empresa, desde una vista principal que administra las empresas afiliadas a una caja de compensación. Las acciones que componen el escenario son extraídas a partir de la descripción realizada en la Tabla 3-3. La especificación de las reglas del modelo para el escenario de adicionar una nueva empresa *(*que se ha denominado *ScenarioAddCompanyModel)* es mostrada en la siguiente estructura de código:

Figura 3-5: Especificación de las reglas del modelo para escenario de ingresar una nueva empresa (*ScenarioAddCompanyModel*), utilizando el Formulario de Empresa.

```
/// <summary>
/// Regla que representa la acción para abrir la interfaz principal
/// que contiene al formulario de empresa
/// </summary>
[Rule]
public string LaunchView_1()
{
    return _mainManager.LaunchView_1();
}

/// <summary>
/// Regla que representa la acción de cerrar la interfaz principal
/// que contiene al formulario de empresa.
/// </summary>
[Rule]
public string Close_1()
{
    return _mainManager.Close_1();
}

/// <summary>
/// Regla que representa la acción de abrir el Formulario de Empresa
/// </summary>
[Rule]
public string AddNewCompany_1()
{
    return _mainManager.AddNewCompany_1();
}
```

```
/// <summary>
/// Regla que representa la acción de ingresar datos al formulario de empresa.
/// </summary>
/// <returns></returns>
[Rule]
public string CompanyViewScenario_1(
        string nit, string nombre,
        string representanteLegal,
        string direccion, string telefono,
        string fechaVinculacion)
{
 return _mainManager.CompanyViewScenario_1(
                                                nit,
                                                nombre,
                                                representanteLegal,
                                                direccion,
                                                telefono,
                                                fechaVinculacion);
}

/// <summary>
/// Regla que representa la acción de guardar los datos de la empresa.
/// </summary>
/// <returns></returns>
[Rule]
public string Guardar_Company_1()
{
return _mainManager.Guardar_Company_1();
}

/// <summary>
/// Regla que representa la acción de cerrar el cuadro de mensaje que informa al
/// usuario que el proceso de Guardar los datos de la empresa fue satisfactorio.
/// </summary>
 [Rule]
public string MsgAckProceesOk_Company_1()
{
return _mainManager.MsgAckProceesOk_Company_1();
}

/// <summary>
/// Acción de cerrar el Formulario de Empresa.
/// </summary>
[Rule]
public string Cancel_Company_1()
{
return _mainManager.Cancel_Company_1();
}
```

Como lo muestra la estructura de código anterior, las reglas definidas en el *ScenarioAddCompanyModel* invocan las acciones que expone el *MainManagerModel* (de acuerdo al esquema de la Figura 3-4**)**. En el *MainManagerModel* se especifican las precondiciones y poscondiciones que precisan el comportamiento deseado de la interfaz. Con la herramienta las precondiciones se definen con la cláusula *Condition* y las poscondiciones indican las acciones que se deben invocar en la implementación, éstas

se implementan en código c# soportándose en la API de *Microsoft.Modeling* provista por la herramienta. Este elemento también expone las propiedades que definen el comportamiento de la interfaz (ver Tabla 3-3), las cuales son observadas luego de la ejecución de cada acción. La especificación del *MainManagerModel* para el Formulario de Empresa es mostrada en la siguiente estructura de código.

Figura 3-6: Especificación del *MainManagerModel*. Se especifican las propiedades que describen el estado de la interfaz y las precondiciones y poscondiciones de las acciones que componen este objeto del modelo.

a) Acción de abrir e inicializar la interfaz principal.

```
/// <summary>
/// Acción para abrir la interfaz principal con ciertos valores de prueba en la lista de empresas.
/// </summary>
public void LaunchView()
{
/// Precondición: La interfaz principal no debe estar abierta.
Condition.IsTrue(!WindowManager.WindowManager.Instance.IsOpen(TipoWindow.MainEmployersManager));
/// La lista de empresas se inicializa con valores de prueba.
_companias.Add("12345678", new CompanyModel(12345678, "Fither S.A", "Tomas Eddisson",
                   "Cr 1 cll 54", "2345678", DateTime.Today));
_companias.Add("87654321", new CompanyModel(87654321, "Active", "Leonardo Puerta",
                  "Cr 2 Av 40", "2111111", DateTime.Today));
/// Poscondición: Se debe ejecutar la operación de iniciar la vista principal por medio del
/// presenter que la maneja.
 _mainPresenter.Iniciar();

}
```

b) Acción de abrir el formulario de empresa.

```
/// <summary>
/// Acción para abrir el formulario de Empresa.
/// </summary>
public void AddNewCompany()
{
/// Precondición: El formulario de empresa no debe estar activo.
Condition.IsTrue(!WindowManager.WindowManager.Instance.IsOpen(TipoWindow.Company));
/// Poscondición: Se invoca la acción de abrir el formulario de empresa por medio del /// presenter
que lo maneja.
_mainPresenter.PresenterCompanyModel.Iniciar("");
}
```

c) Acción de ingresar datos al formulario de empresa.

```
/// <summary>
/// Acción que emula el ingreso de datos en el formulario de empresa.
/// </summary>
public void CompanyViewScenario(string nit, string nombre, string representanteLegal, string
direccion, string telefono, string fechaVinculacion)
{
/// Precondición: El formulario de empresa debe estar activo
Condition.IsTrue(WindowManager.WindowManager.Instance.IsEnabled(TipoWindow.Company));
/// Poscondición: El formulario de empresa debe tener los datos ingresados.
_mainPresenter.PresenterCompanyModel.Vista.Nit = nit;
```

```
_mainPresenter.PresenterCompanyModel.Vista.Nombre = nombre;
_mainPresenter.PresenterCompanyModel.Vista.RepresentanteLegal = representanteLegal;
_mainPresenter.PresenterCompanyModel.Vista.Direccion = direccion;
_mainPresenter.PresenterCompanyModel.Vista.Telefono = telefono;
_mainPresenter.PresenterCompanyModel.Vista.FechaVinculacion = fechaVinculacion;
/// Poscondición: Se debe invocar la acción de validar los datos ingresados en el formulario de
/// empresa, por medio del presenter que lo maneja.
_mainPresenter.PresenterCompanyModel.ViewValidarInterfaz();
}
```

d) Acción de oprimir el botón de guardar del formulario de empresa.

```
/// <summary>
/// Acción base de guardar los datos de una empresa.
/// Emula la acción de oprimir el botón de guardar.
/// </summary>
public void Guardar_Company()
{
/// Precondición: El formulario de empresa debe estar activo.
Condition.IsTrue(WindowManager.WindowManager.Instance.IsEnabled(TipoWindow.Company));
/// Precondición: El message box de proceso satisfactorio no debe estar activo.
Condition.IsTrue(!WindowManager.WindowManager.Instance.IsOpen(TipoWindow.MsgProcessOK));
/// Precondición: El formulario de empresa debe tener un estado válido para guardar.
Condition.IsTrue(_mainPresenter.PresenterCompanyModel.Vista.HabilitarGuardar);
/// Poscondición: Se debe invocar la acción de guardar del formulario de empresa por
/// medio del presenter que lo maneja.
_mainPresenter.PresenterCompanyModel.Guardar();
}
```

e) Acción de cerrar el cuadro de mensaje informativo que indica que el proceso de guardar se realizó satisfactoriamente.

```
/// <summary>
/// Acción para cerrar el cuadro de mensaje que indica que el proceso fue satisfactorio.
/// </summary>
public void MsgAckProceesOk_Company()
{
/// Precondición: El cuadro de mensaje informativo debe estar activo.
Condition.IsTrue(WindowManager.WindowManager.Instance.IsEnabled(TipoWindow.MsgProcessOK));
/// Poscondición: Se invoca la acción de cerrar el cuadro de mensaje informativo.
_mainPresenter.PresenterCompanyModel.Vista.CloseProcesoOk();
}
```

f) Acción de cerrar el formulario de empresa.

```
/// <summary>
/// Acción base de cancelar el formulario de empresa.
/// </summary>
public void Cancel_Company()
{
/// Precondición: el formulario de empresa debe estar activo
Condition.IsTrue(WindowManager.WindowManager.Instance.IsEnabled(TipoWindow.Company));
/// Precondición: el message box de proceso satisfactorio no debe estar activo
Condition.IsTrue(!WindowManager.WindowManager.Instance.IsOpen(TipoWindow.MsgProcessOK));
/// Poscondición: Se cierra el formulario de empresa por medio del presenter que lo
/// maneja.
_mainPresenter.PresenterCompanyModel.Close();
}
```

g) Acción de cerrar la interfaz principal.

```
/// <summary>
/// Acción para cerrar la interfaz principal.
/// </summary>
public void Close()
{
/// Precondición: La interfaz principal debe estar abierta.
Condition.IsTrue(WindowManager.WindowManager.Instance.IsEnabled(TipoWindow.MainEmployersManager));
/// Poscondición: Se invoca la acción de cerrar la vista principal por medio del presenter que la
/// maneja.
_mainPresenter.Close();
}
```

h) Propiedades que describen el estado del formulario de empresa.

```
/// <summary>
/// Estado del Formulario de Empresa.
/// Contiene el consolidado de las propiedades definidas para modelar.
/// </summary>
public string EstadoInterFazCompany
{
  get
  {
    return
     "IsValid:"  + _mainPresenter.PresenterCompanyModel.Vista.IsValid.ToString()+
     "IsDirty: " + _mainPresenter.PresenterCompanyModel.Vista.IsDirty.ToString()+
     "HabilitarGuardar:"+_mainPresenter.PresenterCompanyModel.Vista.HabilitarGuardar.ToString()+
     "ActiveWindow: " + ActiveWindow +
     "Items:" + _companias.Count.ToString();
    }
}
/// <summary>
/// Indica los campos que poseen errores de validación
/// </summary>
public string CamposConErrores
{
   get
   {
    return
   "CampoConErrores:"+_mainPresenter.PresenterCompanyModel.Vista.CamposConErrores;
   }
}
/// <summary>
/// Indica la ventana activa consultando el window manager (un objeto del Helper).
/// </summary>
private string ActiveWindow
{
   get
   {
    if (WindowManager.WindowManager.Instance.IsEnabled(TipoWindow.MsgProcessOK))
      _activeWindow = "MsgProcessOK";
    else if (WindowManager.WindowManager.Instance.IsEnabled(TipoWindow.Company))
      _activeWindow = "Company";
    else if (!WindowManager.WindowManager.Instance.IsOpen(TipoWindow.MainEmployersManager))
     _activeWindow = "NotOpen";
    else _activeWindow = "EmployersManager";
    return _activeWindow;
```

El siguiente paso es configurar desde Spec Explorer el modelo, *ScenarioAddCompanyModel* (Figura 3-5)**,** que indica cómo éste será transformado a una máquina de estados finita. Se realiza la configuración de:

- ✓ Las acciones que serán probadas desde la implementación (hay una correspondencia entre las reglas definidas en el *ScenarioAddCompanyModel* y las operaciones del componente de la implementación).
- ✓ Valores de prueba en los datos de entrada del Formulario de Empresa.
- ✓ El escenario a partir del cual se explorará el modelo (la secuencia explícita de operaciones que se ejecutarán sobre la interfaz).

A continuación se presenta la configuración realizada para el modelo: ScenarioAddCompanyModel, la cual dirige la manera como se genera la FSM.

Figura 3-7: Configuración realizada para el Modelo: *ScenarioAddCompanyModel.*

```
/// 1. Acciones que serán probadas desde la implementación.
config MainAddNewCompany
{
action ScenarioAddCompanyImplementation();
action  string  ScenarioAddCompanyImplementation.LaunchView_1();
action  string  ScenarioAddCompanyImplementation.AddNewCompany_1();
action  string  ScenarioAddCompanyImplementation.CompanyViewScenario_1(string nit,
string nombre, string representanteLegal, string direccion, string telefono, string
fechaVinculacion);
action  string  ScenarioAddCompanyImplementation.MsgAckProceesOk_Company_1();
action  string  ScenarioAddCompanyImplementation.Guardar_Company_1();
action  string  ScenarioAddCompanyImplementation.Cancel_Company_1();
action  string  ScenarioAddCompanyImplementation.Close_1();
}
/// 2. Valores de prueba. Para cada campo se configuran valores válidos e
inválidos.
config ParameterCombinationAddNewCompany: MainAddNewCompany
{
action  string  ScenarioAddCompanyImplementation.CompanyViewScenario_1
(string nit, string nombre, string representanteLegal, string direccion, string
telefono, string fechaVinculacion)

where
{.
Condition.In(nit,"12345678", "999999999999999","1000000000000000", "letras", "-1" );
Condition.In(nombre, "AYAX LTDA.", " ");
Condition.In(representanteLegal, "Tomas Edisson", "xxxxxxxxxxxxxxxxxxxx");
Condition.In(direccion, "Cr 5 Av 4", " ");
Condition.In(telefono, "2233445", " ");
Condition.In(fechaVinculacion, "2012-01-31", "2012-12-31");
.};
}
```

```
/// 3. Definición del escenario de prueba.

machine AddNewCompanyScenario() : MainAddNewCompany where ForExploration = true
{
(new ScenarioAddCompanyImplementation; LaunchView_1; AddNewCompany_1;
CompanyViewScenario_1 ; Guardar_Company_1 ; MsgAckProceesOk_Company_1;
Cancel_Company_1;Close_1)
}
```

Al explorar el modelo con la herramienta, se genera una máquina de estados con la cual se puede analizar visualmente el comportamiento de éste, pues despliega los posibles estados y transiciones según la combinación de todos los valores de prueba establecidos. Con los valores de entrada definidos en la Figura 3-7 se genera una FSM de 804 estados y 803 transiciones. Por fines prácticos y para visualizar mejor los diagramas resultantes, se mostrará la FSM generada con un dominio pequeño de valores de prueba. En las siguientes figuras se presentan los diagramas generados para distintos valores de prueba y validando las propiedades descritas en la Tabla 3-3.

Figura 3-8: FSM generada cuando el Nit se ingresa con un valor invalido (-1) y un valor válido (1); los demás datos se ingresan con valores válidos. Se observa el resultado de las propiedades HabilitarGuardar e Items.

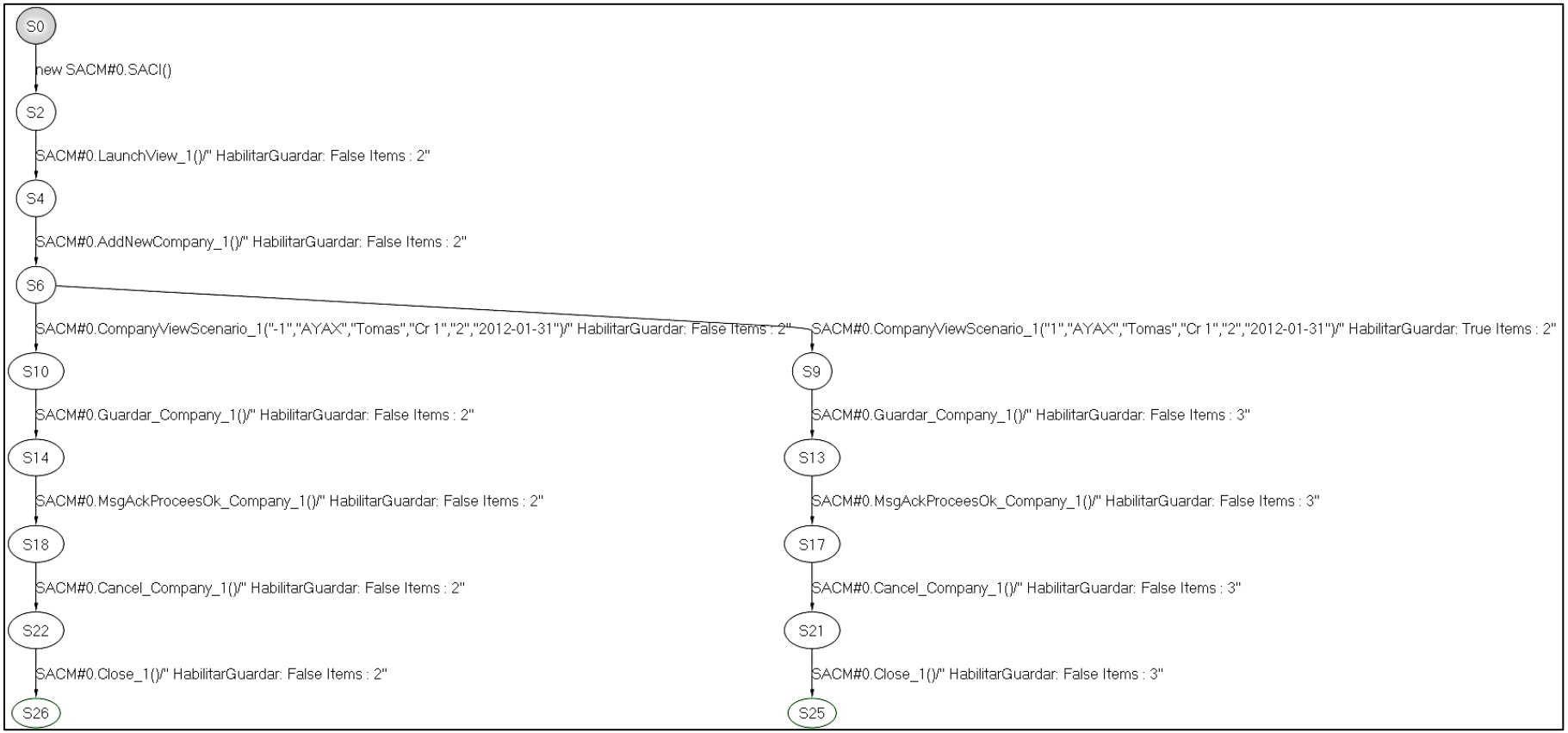


Figura 3-9: FSM generada cuando el Nit se ingresa con un valor invalido (-1) y un valor válido (1); los demás datos se ingresan con valores válidos. Se observa el resultado de la propiedad ActiveWindow.

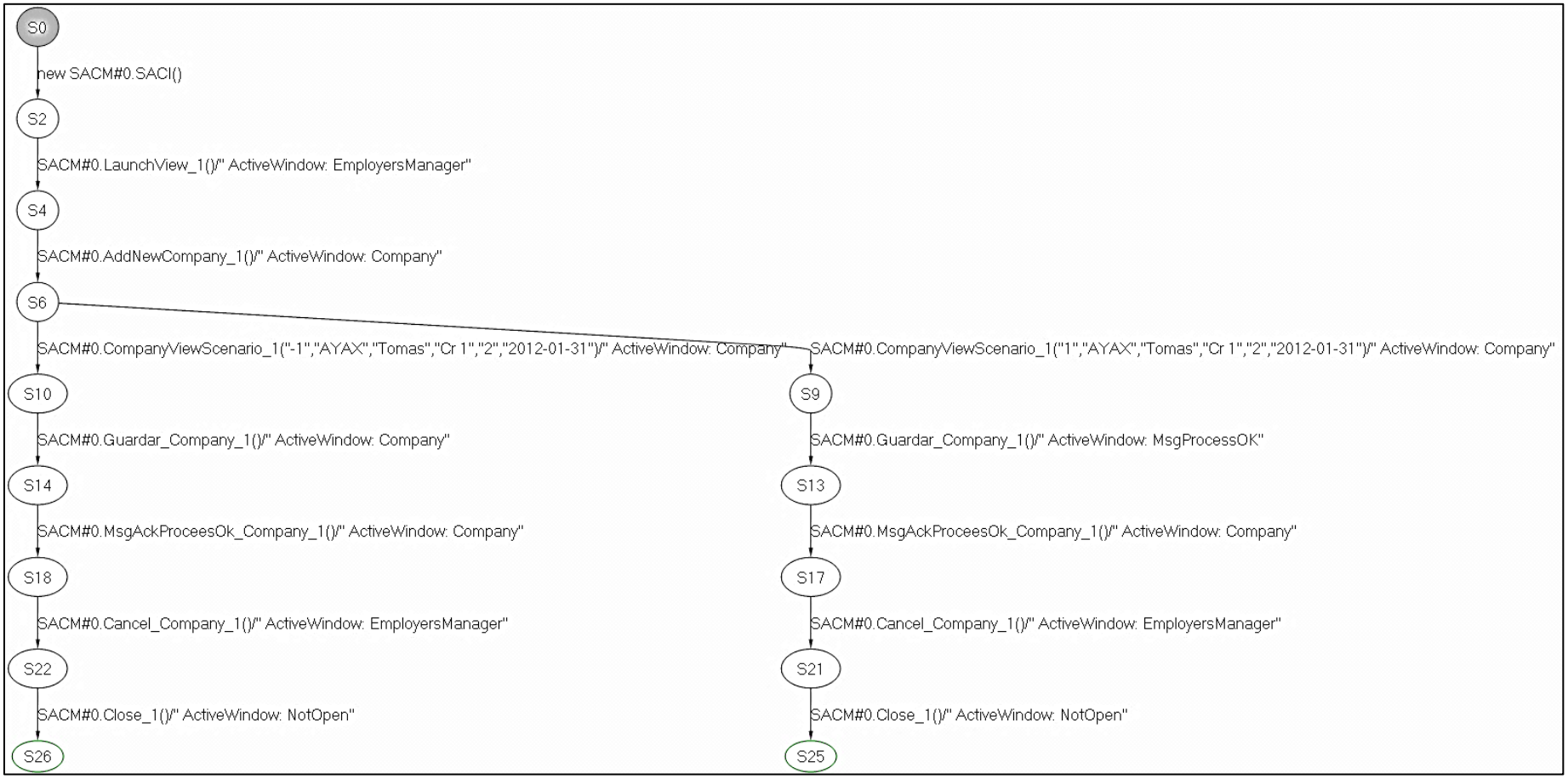


Figura 3-10: FSM generada cuando el Nombre se ingresa con un valor invalido (“”) y un valor válido (AYAX); los demás datos se ingresan con valores válidos. Se observa el resultado de la propiedad CampoConErrores.

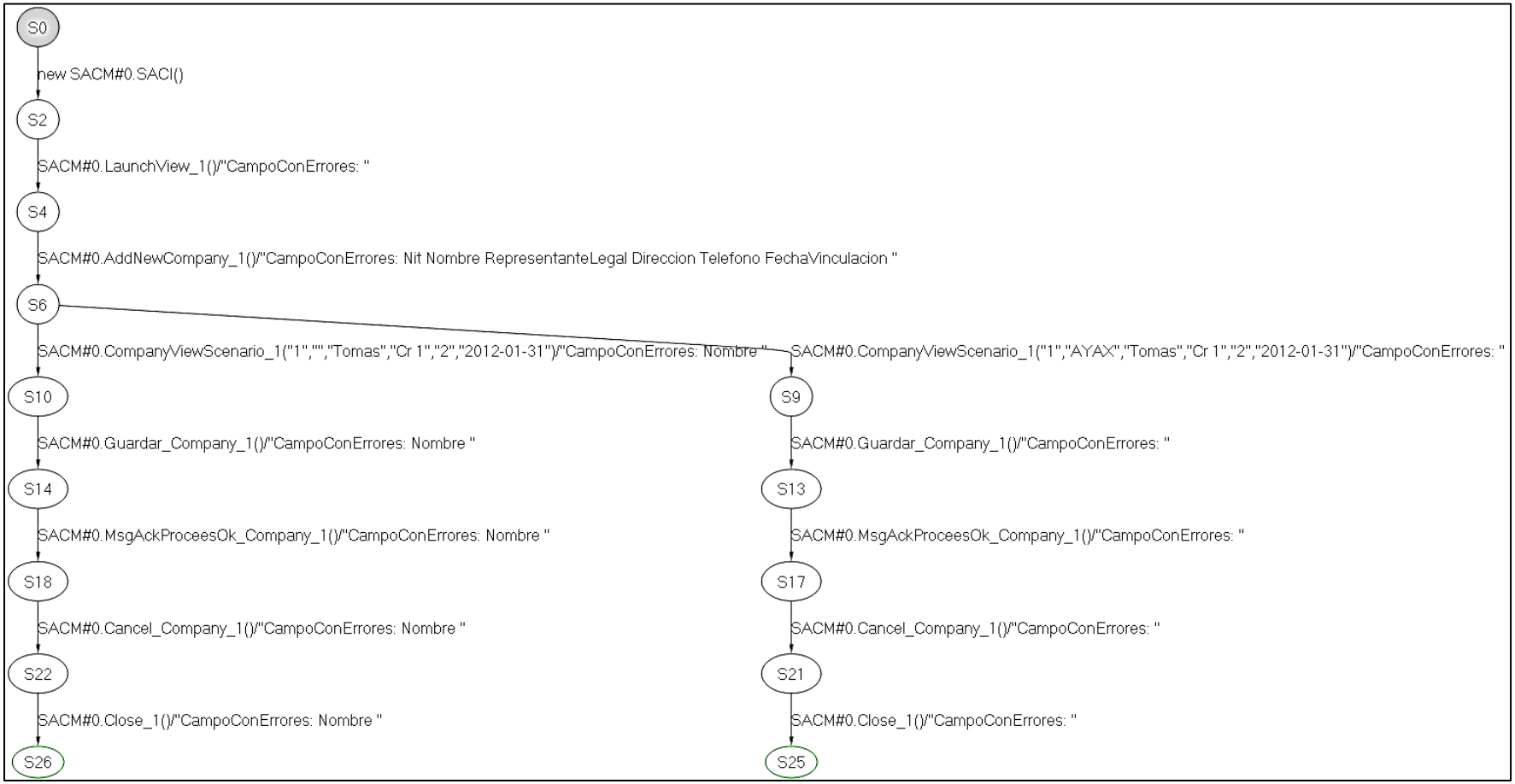
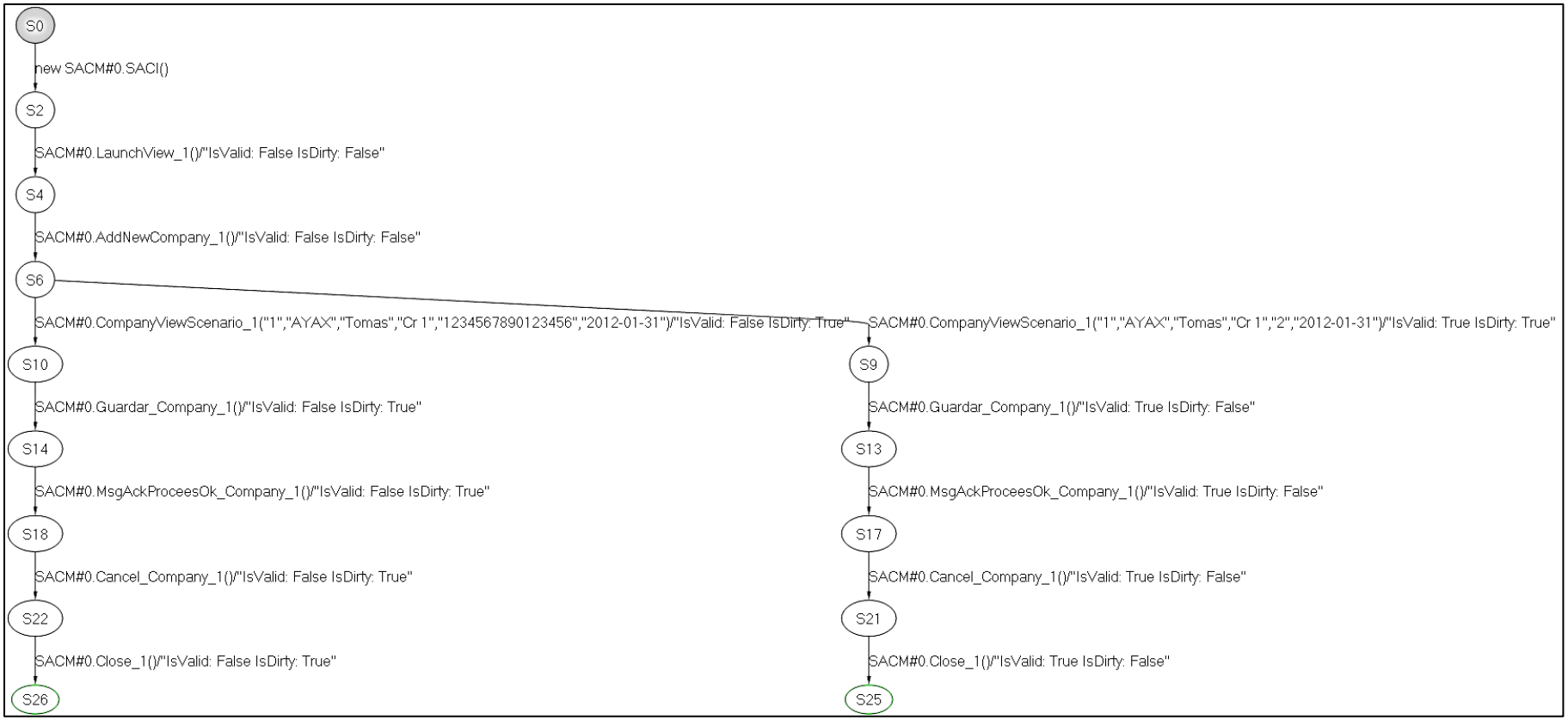


Figura 3-11: FSM generada cuando el Teléfono se ingresa con un valor invalido (1234567890123456) y un valor válido (2); los demás datos se ingresan con valores válidos. Se observa el resultado de las propiedades IsValid e IsDirty.



3.3 Automatización de la Generación de Casos de Prueba

La automatización se logra con la herramienta al generar de los casos de prueba a partir de dos elementos fundamentales: la FSM construida para el modelo, que representa el comportamiento deseado de la interfaz, y del criterio de cobertura seleccionado para la generación de los casos de prueba. Con Spec Explorer a partir de la FSM se generan los casos de prueba abstractos o de alto nivel, que son una representación lógica de los escenarios de prueba definidos en el modelo, y a partir de estos casos se generan las pruebas concretas que posteriormente se ejecutan sobre la implementación de la GUI, obteniendo un reporte de resultados de la ejecución de la suite de pruebas.

3.3.1 Criterio de Selección de Pruebas

En MBT el criterio de selección de pruebas se refiere al análisis para la generación de la suite de pruebas con la cual se pueda validar el comportamiento de la GUI. Los criterios de selección de pruebas están relacionados a los modelos y a los requerimientos; es decir, miden qué tan bien la suite de pruebas generada cubre el modelo y qué tan bien la implementación esta acoplada con el modelo [1]. La selección de pruebas se basa en los criterios de cobertura para controlar el diseño y la generación de la suite de pruebas [1, 26].

Los criterios de cobertura aplicados en el desarrollo del trabajo están basados en las capacidades brindadas por la herramienta (*Spec Explorer 2010,* en [26] se puede consultar una revisión de los criterios que soporta*)* y los objetivos de pruebas que se desean cumplir, ya que a partir de esto se determina los algoritmos que ésta usa para generar la suite de pruebas, lo que implica la cantidad de pruebas que se generarán, el tiempo de generación y las partes de la implementación que serán probadas.

Los criterios de cobertura soportados por la herramienta y por ende aplicados en el desarrollo del trabajo son (basados en [1] y en [26]):

- ✓ Criterio de cobertura estructural del modelo: Se aplica por medio de los siguientes criterios:
  - *Interface/Function Criteria:* Se refiere a la cobertura que poseen las funciones definidas en el modelo dentro de la implementación, por medio de la suite de pruebas generada. En *Spec Explorer* este criterio se cubre totalmente ya que obliga a mapear las reglas de las acciones del modelo en la implementación, es decir todas las acciones presentes en los *Scenarios Models* tienen una relación uno a uno con las operaciones de los *Scenarios Implementations*.
  - *Statement Coverage Criteria:* Un set de pruebas alcanza esta cobertura cuando cada línea de código de la implementación es probado al menos una vez. Se medirá cuantitativamente directamente sobre la implementación por medio del *Code Coverage Result* de las pruebas.
  - *All-transitions coverage:* Se alcanza si cada transición del modelo es atravesado al menos una vez. *Spec Explorer* cubre totalmente este criterio, ya que genera una máquina de estados que explora explícitamente el modelo y todas las transiciones definidas en la FSM del modelo son generadas en la suite de pruebas.
- ✓ Criterio de cobertura de datos: Se refiere a la cobertura del domino de los datos de entrada de una operación o transición en el modelo. Se emplea para seleccionar un adecuado conjunto de valores como entrada de los datos de las pruebas cuando existe un gran número de posibles valores de entrada. Se aplica por medio del siguiente parámetro:
  - *Boundary value coverage:* Para restricciones en valores de entrada, este criterio se alcanza si al menos un punto de cada frontera es probado [26]. Se aplicará configurando, desde Spec Explorer, las acciones que tienen parámetros de entrada con valores dentro de las fronteras y fuera de éstas, de acuerdo a los requerimientos funcionales, por ejemplo si un campo de un formulario debe ser un número mayor o igual a cero de máximo 5 caracteres, los valores escogidos serian: valores de frontera válidos (“0” y ‘’99999’’), valores fuera de las fronteras inválidos (‘’letras’’ y “-1”). Se medirá por el número de estados, transiciones y casos de pruebas generados por la herramienta dependiendo de la combinación de los valores de entrada.
- ✓ Especificaciones de casos de prueba explícitos: Este permite realizar una validación explicita de un determinado requerimiento generando un conjunto de pruebas

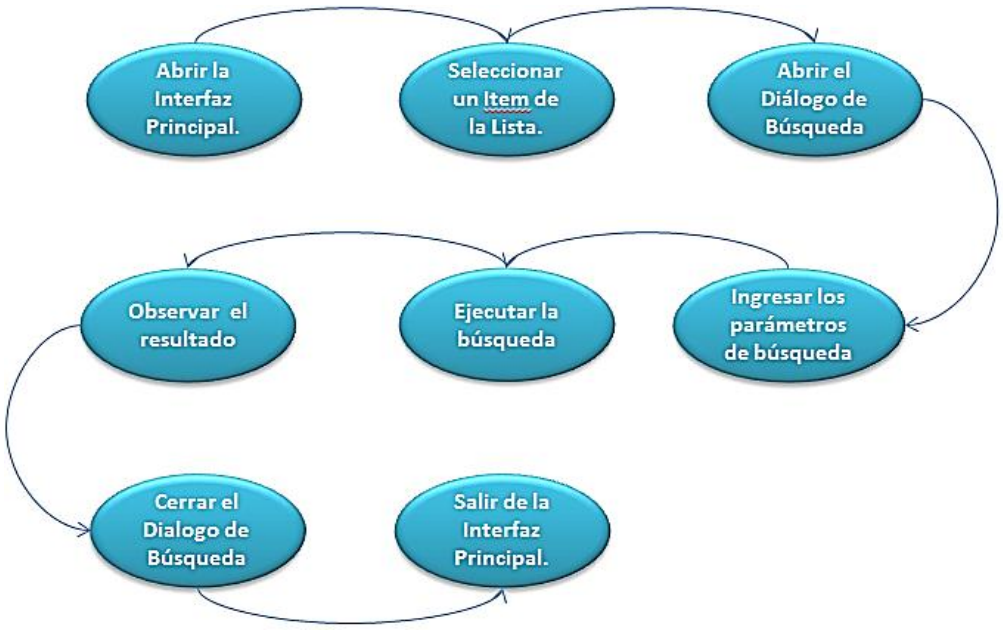
específicas del modelo. Este se aplicará tomando el enfoque de abstraer los escenarios de los casos de pruebas en modelos independientes, de tal forma que el modelo general sea una composición de sub-modelos que representen los escenarios de prueba principales, de esta manera se logra crear un modelado más comprensible.

3.3.2 Desarrollo de los Casos de Prueba Abstractos

Los casos de prueba abstractos o de alto nivel son una representación lógica de los escenarios de pruebas en términos del modelo. En el contexto de este trabajo definiremos un escenario como la simulación de una secuencia de pasos o acciones que un usuario realizaría para emplear una funcionalidad determinada de la GUI y el cual debe ser independiente del estado de otros escenarios. El modelado de estos escenarios se define precisamente en los *Scenarios Models* mostrados en la Figura 3-4.

Un ejemplo de la especificación de un escenario de prueba es mostrado en la Figura 3-12**,** en donde un usuario realiza un proceso de búsqueda luego de ingresar a la interfaz.

Figura 3-12: Ejemplo de la especificación de un escenario para un diálogo de búsqueda.



Los casos de prueba abstractos se generan luego de haber definido:

- ✓ La FSM que modela los escenarios.
- ✓ Los valores de los datos de prueba. Con base en el criterio de cobertura seleccionado (*Boundary value coverage*).

En la Tabla 3-4 se presentan los valores de pruebas seleccionados para el ejemplo del Formulario de Empresa, de acuerdo a las reglas de validación definidas en la Tabla 3-2 para los campos de entrada que componen el formulario y al criterio de cobertura de datos (*Boundary value coverage*), descrito en la sección anterior.

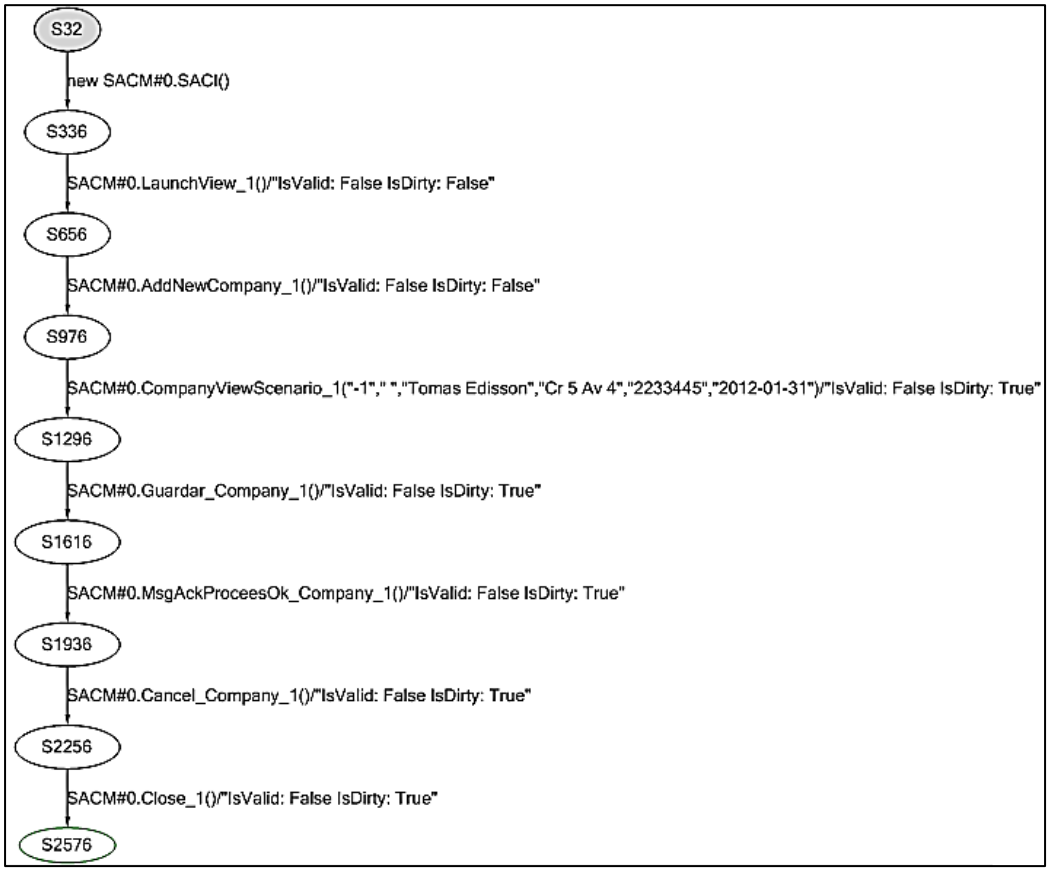
Tabla 3-4: Descripción de los valores de prueba para el escenario de agregar una empresa utilizando el Formulario de Empresa.

Acción	Parámetro de Entrada	Valores de Prueba	Descripción
LaunchView	--	--	Se activa la interfaz principal.
AddNewCompany	--	--	Se activa el formulario de empresa.
CompanyView Scenario	Nit	"12345678"	Valor válido, esta dentro del rango (>= 0 y <= 99999999999999).
		"999999999999999"	Valor de frontera válido (= 99999999999999).
		"1000000000000000",	Valor inválido, esta fuera del rango (>= 0 y <= 99999999999999).
		"letras",	Valor inválido, no es un número.
		"-1"	Valor inválido, esta fuera del rango (>= 0 y <= 99999999999999).
	Nombre	""AYAX LTDA."	Valor válido, es un texto de menos de 20 caracteres de longitud.
		" "	Valor inválido, es un texto vacío.
	Representante Legal	"Tomas Edisson"	Valor válido, es un texto de menos de 20 caracteres de longitud.
		"xxxxxxxxxxxxxxxxxxxx"	Valor inválido, es un texto de más de 20 caracteres de longitud.
	Teléfono	"2233445"	Valor válido, es un texto de menos de 20 caracteres de longitud.
		" "	Valor inválido, es un texto vacío.
	Dirección	"Cr 5 Av 4"	Valor válido, es un texto de menos de 20 caracteres de longitud.
		" "	Valor inválido, es un texto vacío.
	Fecha Vinculación	"2012-01-31"	Fecha válida, es menor a la afecha actual.
		"2012-12-31"	Fecha inválida, es mayor a la afecha actual.
Guardar	--	--	Se actualiza la lista principal.
MsgAckProceesOk	--	--	Se activa un cuadro de mensaje informativo.
Cancel	--	--	Se desactiva el formulario de empresa.
Close	--	--	Se desactiva la interfaz principal.

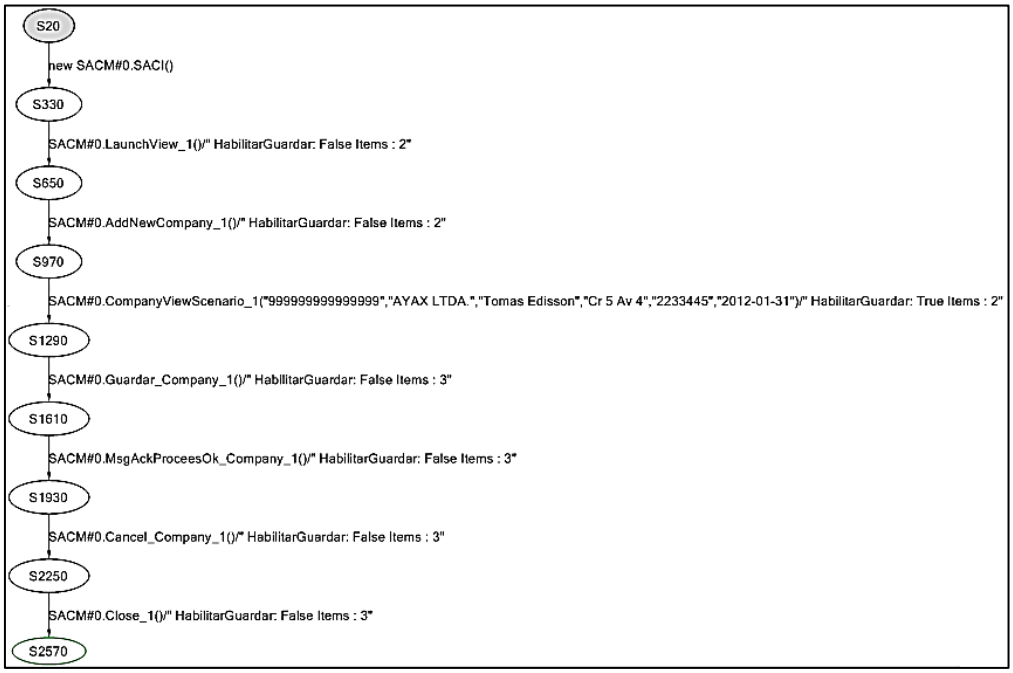
Con la definición de los datos prueba y la FSM de los escenarios se generan los casos de prueba abstractos. La herramienta realiza una generación con la combinación de todos los valores especificados y el resultado esperado es obtenido del comportamiento del modelo diseñado. Para el caso del Formulario de Empresa se generaron 160 casos de prueba de alto nivel en donde cada uno realiza las acciones del escenario pero con diferentes valores de prueba. En la Figura 3-13 se presentan algunos casos de prueba de alto nivel que se generaron automáticamente luego de especificar los escenarios, el modelo y los datos de prueba para el Formulario de Empresa. Para cada caso de prueba se muestran los valores de prueba de entrada y los resultados esperados de las propiedades modeladas (Tabla 3-3).

Figura 3-13: Ejemplos de casos de prueba de alto nivel generados a partir del modelado del escenario especificado para el Formulario de Empresa.

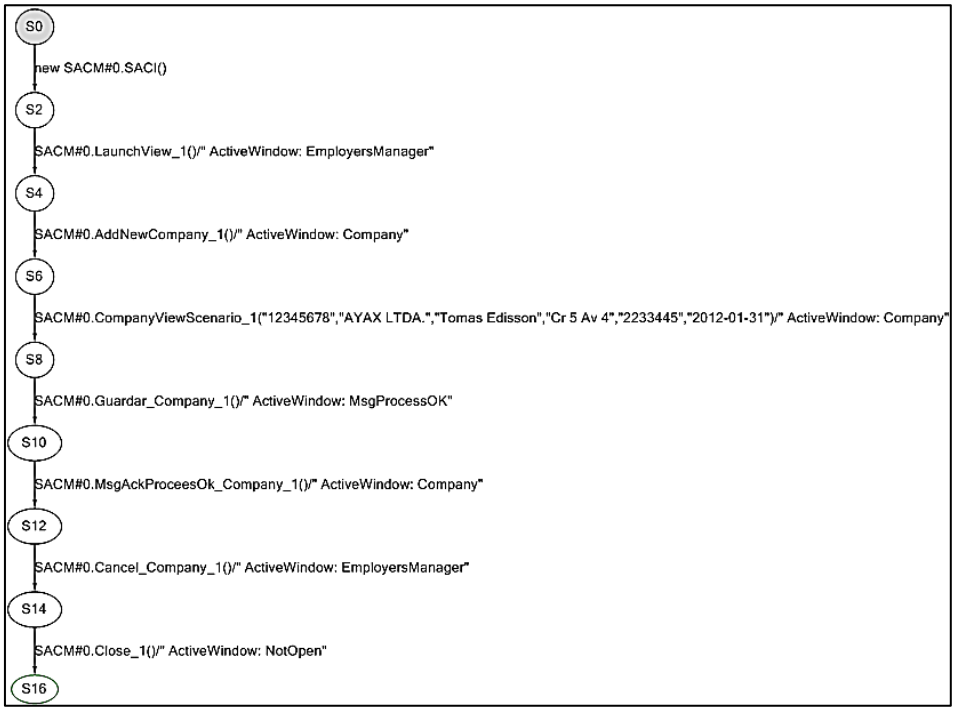
a) Caso de prueba ingresando datos inválidos en Nit y Nombre. Se observa el resultado de las propiedades IsValid e IsDirty.



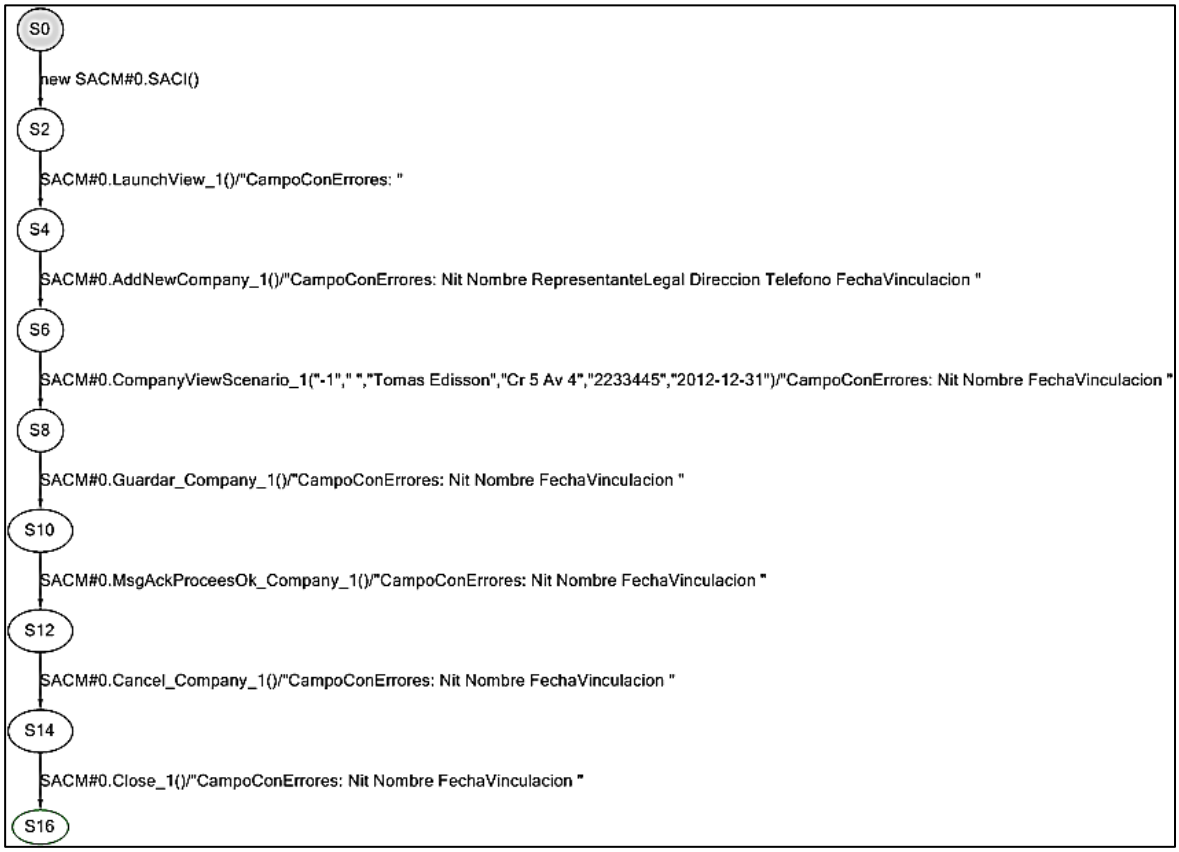
a) Caso de prueba ingresando datos válidos. Se observa el resultado de las propiedades HabilitarGuardar e Items.



a) Caso de prueba ingresando datos válidos. Se observa el resultado de la propiedad ActiveWindow.



a) Caso de prueba ingresando datos inválidos en Nit, Nombre y Fecha. Se observa el resultado de la propiedad CamposConErrores.



3.3.3 Generación de los Casos de Prueba Ejecutables

La última parte del proceso es generar el código ejecutable de pruebas para poder probar la implementación y verificar que el comportamiento de la interfaz gráfica de usuario está acoplado con el comportamiento esperado dado por el modelo creado. En este punto se diseña y se construye la GUI concreta, hasta que todas las pruebas, generadas a parir del modelo, pasen. El código de pruebas ejecutado sobre la implementación encontrará errores si la GUI no está cumpliendo con los escenarios modelados, que son una abstracción de los requerimientos definidos.

A partir de los casos de prueba de alto nivel la herramienta genera los casos de pruebas ejecutables, que apuntan directamente a la implementación; en este caso el código de pruebas generado automáticamente se establece en forma de pruebas de unidad de

VSTT (*Visual Studio Team Test*) y prueban explícitamente la implementación de los escenarios o el manager, según sea el caso, si es una interfaz compuesta o simple (Figura 3-14).

Figura 3-14: Casos de pruebas ejecutables generados en componentes Test Suite.

Para el ejemplo que se ha venido detallando (escenario de añadir una empresa nueva empleando el Formulario de Empresa), los casos de prueba ejecutables se generan en el componente *ScenarioAddCompanyModelTestSuite*. Esta suite prueba directamente el objeto de la implementación del escenario, el cual es denominado: *ScenarioAddCompanyImplementation*.

La configuración que se realiza desde la herramienta para generar la suite de pruebas es mostrada a continuación:

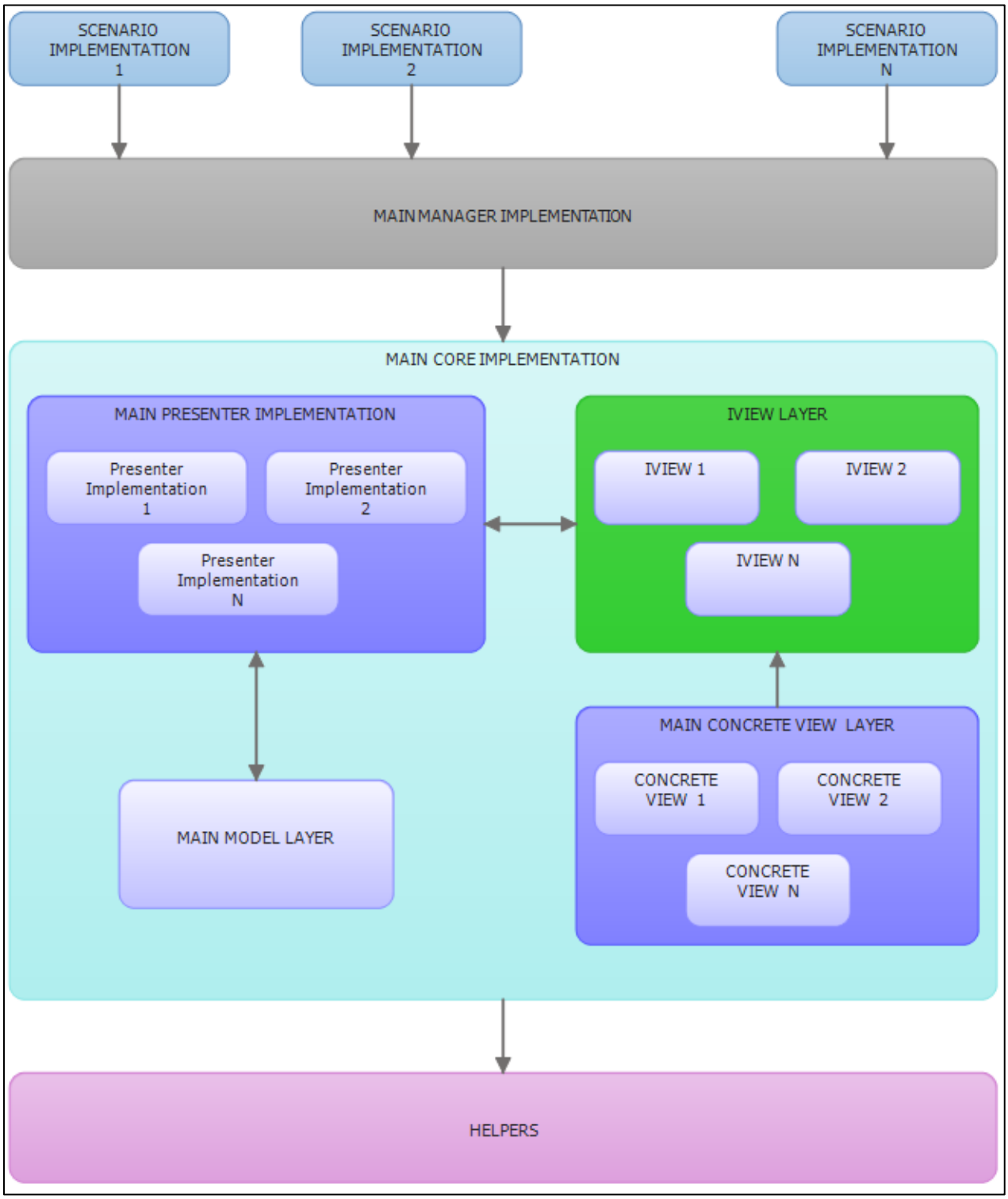
```
/// Configuración de la construcción de la suite de pruebas para el escenario de
/// agregar una empresa nueva empleando el Formulario de Empresa.
machine ScenarioAddCompanyModelTestSuite() : MainAddNewCompany where TestEnabled = true
{
    construct test cases where strategy = "longtests" for SliceAddNewCompanyScenario()
}
```

3.3.3.1 Implementación de la GUI Concreta

Con el enfoque propuesto, la implementación debe tener la misma estructura que la del modelo, ya que al fin de cuentas el modelo es una simplificación de la implementación [1, 24]. En la implementación se crean los componentes concretos presentes en el modelo que finalmente acceden, activan y manipulan la interfaz gráfica de usuario real para luego poder realizar la ejecución automática de los casos de prueba.

En la Figura 3-15 se presenta la arquitectura concreta empleada para aplicar el proceso de pruebas basadas en modelos sobre interfaces gráficas de usuario compuestas. Al igual que en la fase de modelado existen los objetos para los escenarios (*Scenarios Implementation*), el manejador principal (*Main Manager Implementation*), el presentador principal (*Main Presenter Implementation*) y se desarrollan las interfaces gráficas de usuario o vistas concretas en alguna tecnología (*Main Concrete View Layer*), implementando las interfaces definidas en la fase del modelado (IView Layer). Para el desarrollo de este trabajo las interfaces gráficas de usuario concretas se implementaron en *Windows Forms*, que es una tecnología de Microsoft para desarrollar aplicaciones desktop (Anexos B, C y D).

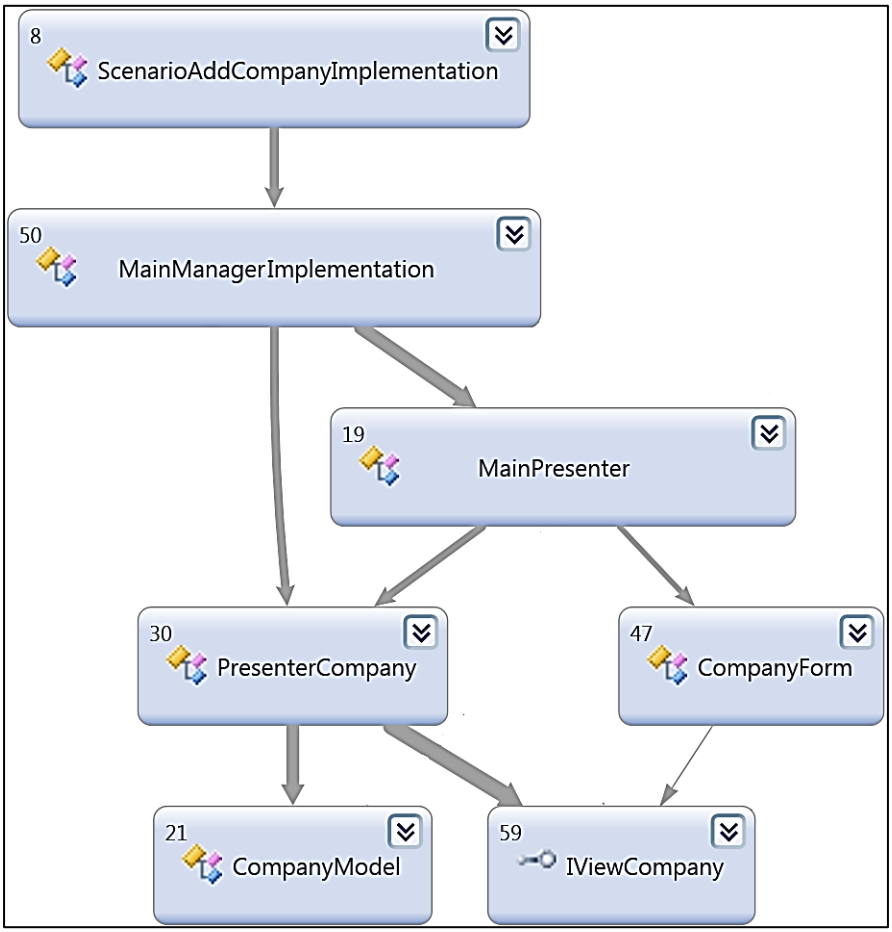
Figura 3-15: Arquitectura concreta para la implementación del proceso MBT sobre GUIs.



Todos los componentes que participan en la estructura anterior poseen las mismas responsabilidades y funciones que se describieron en la generación del modelo, pero aplicados a la implementación, y también las capas superiores se apoyan en las utilidades de los helpers para poder cumplir con los objetivos de pruebas.

En la Figura 3-16 se presenta el diagrama de dependencias del diseño concreto para el ejemplo de la Figura 3-1, tiene la misma estructura del modelo pero con los detalles concretos de implementación. Si la interfaz esta acoplada con el modelo, quiere decir que cumple con los requerimientos del escenario.

Figura 3-16: Diseño de la implementación concreta para el escenario que emplea el *Formulario de Empresa*.



La descripción de los objetos concretos que componen el diagrama de la implementación (Figura 3-16) es:

- ✓ ***ScenarioAddCompanyImplementation***: Contiene las acciones que componen el escenario de agregar una nueva compañía empleando el Formulario de Empresa. Las acciones tienen la misma estructura de las reglas definidas en el *ScenarioAddCompanyModel*, sino que llaman a las funcionalidades concretas del *MainManagerImplementation*.
- ✓ ***MainManagerImplementation***: Fachada para acceder al resto de funcionalidades. Las operaciones que lo componen tienen la misma estructura a las definidas en el *MainManagerModel*, sino que llaman a los objetos concretos.
- ✓ ***MainPresenterImplementation:*** Maneja el comportamiento de la vista principal, por medio de la cual se invoca el Formulario de Empresa. Contiene el PresenterCompany que gestiona este formulario.
- ✓ ***PresenterCompany***: Maneja el comportamiento del formulario de empresa, por el cual se adicionan los datos de la nueva compañía.
- ✓ ***IViewCompany:*** Interface que expone las funcionalidades y atributos que debe tener el Formulario de Empresa. Por medio de esta interface se comunica el *PresenterCompany* y el Formulario de Empresa *(*denominado *CompanyForm).*
- ✓ ***CompanyForm:*** Es el Formulario de Empresa concreto. Esta clase debe implementar la interface *IViewCompany*.
- ✓ ***CompanyModel:*** Objeto que encapsula los datos de empresa.

Para hacer más comprensible el ejemplo, en los siguientes fragmentos de código se presentan la estructura de los principales objetos de la implementación (*MainPresenterImplementation, PresenterCompany e IViewCompany*).

Figura 3-17: Estructura de código de los objetos de la implementación del Formulario de Empresa.

a) Especificación de la interface *IViewCompany*.

```
// Interface que expone las operaciones y propiedades del formulario de Empresa.
// El presenter se comunica con la vista por medio de esta interface.
public interface IViewCompany
    {
        // Campos de entrada del Formulario de Empresa.
        string Nit { get; set; }
        string Nombre { get; set; }
        string RepresentanteLegal { get; set; }
        string Direccion { get; set; }
        string Telefono { get; set; }
        DateTime FechaVinculacion { get; set; }

        // Propiedades que componen el estado de la vista.
        bool IsValid { get; set; }
        bool IsDirty { get; set; }
        bool HabilitarGuardar { get; set; }
        List<MensajeValidacion> MensajesDeValidacion { get; set; }

        // Eventos enviados del formulario al presenter para ser gestionados.
        event EventHandler GuardarClick;
        event EventHandler CancelarClick;
        event EventHandler ValidarInterfaz;
        event EventHandler HasChanged;
        event EventHandler CloseProcessOkClick;

         // Acciones realizadas sobre el formulario.
        void Mostrar();
        void Close();
        void ShowValidacion();
        void CloseProcesoOk();
        void ShowProcesoOK();
    }
```

b) Especificación del PresenterCompanyImplementation. Presenter encargado de manejar el formulario de empresa.

```
/// <summary>
/// Interface que expone las funcionalidades del Formulario de Empresa (vista).
/// Es la interface por medio de la cual el presenter se comunica con la vista.
/// </summary>
private IViewCompany _vista;
/// <summary>
/// Lista de empresas existentes. Representa el modelo en esta implementación de MVP.
/// </summary>
private Dictionary<string, CompanyModel> _modeloCompanies;
/// <summary>
/// Constructor del presenter. Éste se suscribe a los eventos disparados por el usuario desde la
/// vista.
/// </summary>e
public PresenterCompanyImplementation(IViewCompany v, Dictionary<string, CompanyModel> m )
{
     this._vista = v;
     this._modeloCompanies = m;
    /**** Events Subcriptions ****/
}
/// <summary>
```

```
/// Inicia las variables principales del presenter y del formulario de empresa.
/// Se encarga de desplegar el formulario de empresa.
/// </summary>
public void Iniciar()
{
    /**** Code Implementation****/
}
/// <summary>
/// El presenter responde al evento de cerrar, re-enviado por la vista, y cierra el formulario.
/// </summary>
public void Close()
{
   /**** Code Implementation****/
}
/// <summary>
/// El presenter responde al evento de guardar, re-enviado por la vista,
/// y ejecuta la acción concreta. Actualiza la vista.
/// </summary>
public void Guardar()
{
   /**** Code Implementation****/
}
// <summary>
/// Si el proceso de guardar se realizó con éxito, el presenter
/// le indica a la vista que despliegue un mensaje informativo.
/// </summary>
public void ShowProcesoOK()
{
   /**** Code Implementation****/
}
/// <summary>
/// Cierra el cuadro de mensaje informativo.
/// </summary>
public void CloseProcesoOK()
{
   /**** Code Implementation****/
}
/// <summary>
/// Cuando la vista ha cambiado, el presentar valida los datos de entrada de la vista.
/// Si hay errores de validación le indica a la vista que despliegue los mensajes
/// explícitos de éstos. Actualiza la vista.
/// </summary>
public void ValiadarVista(IViewCompany _vista)
{
    /**** Code Implementation****/
}
```

c) Especificación del *MainPresenterImplementation*. Presenter principal que maneja el *PresenterCompanyImplementation*.

```
/// <summary>
/// Se encarga de desplegar la vista principal.
/// </summary>
public void Iniciar()
{
   /**** Code Implementation****/
}
/// <summary>
/// Se encarga de invocar al presenter que maneja el formulario de empresa.
/// </summary>
public void ShowAddNewCompanyView()
{
   /**** Code Implementation****/
}
```

```
/// <summary>
/// El presenter principal responde al evento de cerrar, re-enviado por la vista, y cierra la
/// vista principal.
/// </summary>
public void Close()
{
  /**** Code Implementation****/
}
```

3.3.3.2 Ejecución de los Casos de Prueba Concretos

El código de pruebas se ejecuta, de manera automatizada, directamente sobre las operaciones definidas en el *Manager Implementation,* que acceden a la GUI concreta mediante el *Presenter Implementation*. En esta fase de pruebas de software, las pruebas se enfocan en probar que el presentador o presentadores estén realizando bien su trabajo de activación, gestión y actualización de la GUI. El presentador es el encargado de abrir, ejecutar la vista, actualizar la vista con las acciones y datos de entrada definidos en los escenarios, responder a los eventos lanzados por la vista, implementar la lógica de presentación y devolver los resultados al *Manager* para que se comparen con los resultados esperados, que están contenidos en la suite de pruebas generada y que reflejan el comportamiento deseado del modelo.

Un ejemplo de un caso de prueba ejecutable generado para probar la implementación del Formulario de Empresa, utilizado dentro del escenario de creación de una nueva empresa (SACI: ScenarioAddCompanyImplementation), es mostrado en el siguiente fragmento de código:

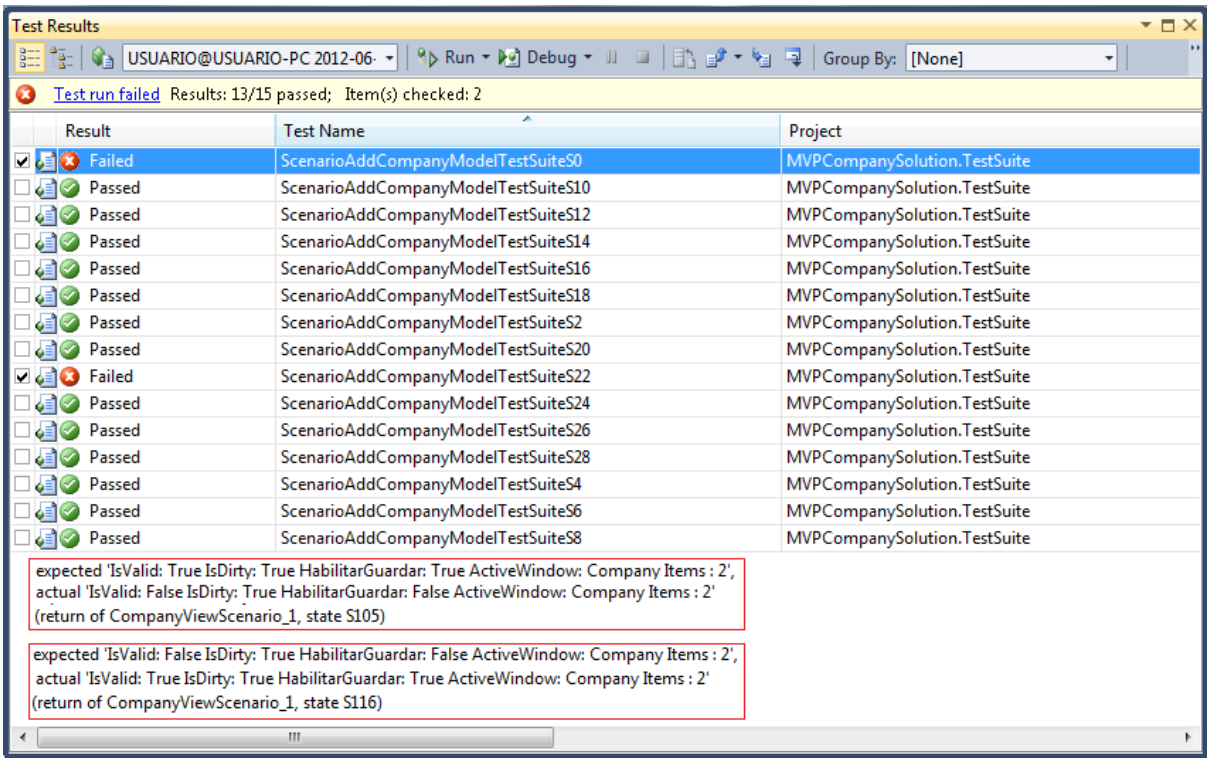
```
#region Test Starting in S0

[Microsoft.VisualStudio.TestTools.UnitTesting.TestMethodAttribute()]

public void ScenarioAddCompanyModelTestSuiteS0()
{

this.Manager.BeginTest("ScenarioAddCompanyModelTestSuiteS0");
this.Manager.Comment("reaching state \'S0\'");
MVPCompanySolution.Sample.SACI temp0;
this.Manager.Comment("executing step \'call new SACM#0.SACI()\'");
temp0 = new MVPCompanySolution.Sample.SACI();
this.Manager.Comment("reaching state \'S1\'");
this.Manager.Comment("checking step \'return new SACM#0.SACI\'");
TestManagerHelpers.AssertBind<MVPCompanySolution.Sample.SACI>(this.Manager, this.o, temp0,
"this of MVPCompanySolution.Sample.SACI, state S1");
this.Manager.Comment("reaching state \'S2\'");
string temp1;
this.Manager.Comment("executing step \'call SACM#0.LaunchView_1()\'");
temp1 = this.o.Value.LaunchView_1();
this.Manager.Comment("reaching state \'S3\'");
this.Manager.Comment("checking step \'return SACM#0.LaunchView_1/\" HabilitarGuardar: False Items :
2\"\'");
```

```
TestManagerHelpers.AssertAreEqual<string>(this.Manager, " HabilitarGuardar: False Items : 2",
temp1, "return of LaunchView_1, state S3");
this.Manager.Comment("reaching state \'S4\'");
string temp2;
this.Manager.Comment("executing step \'call SACM#0.AddNewCompany_1()\'");
temp2 = this.o.Value.AddNewCompany_1();
this.Manager.Comment("reaching state \'S5\'");
this.Manager.Comment("checking step \'return SACM#0.AddNewCompany_1/\" HabilitarGuardar: False
Items : 2\\'");
TestManagerHelpers.AssertAreEqual<string>(this.Manager, " HabilitarGuardar: False Items : 2",
temp2, "return of AddNewCompany_1, state S5");
this.Manager.Comment("reaching state \'S6\'");
string temp3;
this.Manager.Comment("executing step \'call SACM#0.CompanyViewScenario_1(\"1\",\"AYAX
LTDA.\",\"Tomas Edisson\",\"Cr 5 Av 4\",\"2233445\",\"2012-01-31\")\'");
temp3 = this.o.Value.CompanyViewScenario_1("1", "AYAX LTDA.", "Tomas Edisson", "Cr 5 Av 4",
"2233445", "2012-01-31");
this.Manager.Comment("reaching state \'S7\'");
this.Manager.Comment("checking step \'return SACM#0.CompanyViewScenario_1/\" HabilitarGuardar: True
Items: 2\"\'");
TestManagerHelpers.AssertAreEqual<string>(this.Manager, " HabilitarGuardar: True Items : 2", temp3,
"return of CompanyViewScenario_1, state S7");
this.Manager.Comment("reaching state \'S8\'");
string temp4;
this.Manager.Comment("executing step \'call SACM#0.Guardar_Company_1()\'");
temp4 = this.o.Value.Guardar_Company_1();
this.Manager.Comment("reaching state \'S9\'");
this.Manager.Comment("checking step \'return SACM#0.Guardar_Company_1/\" HabilitarGuardar: False
Items : 3\"\'");
TestManagerHelpers.AssertAreEqual<string>(this.Manager, " HabilitarGuardar: False Items : 3",
temp4, "return of Guardar_Company_1, state S9");
this.Manager.Comment("reaching state \'S10\'");
string temp5;
this.Manager.Comment("executing step \'call SACM#0.MsgAckProceesOk_Company_1()\'");
temp5 = this.o.Value.MsgAckProceesOk_Company_1();
this.Manager.Comment("reaching state \'S11\'");
this.Manager.Comment("checking step \'return SACM#0.MsgAckProceesOk_Company_1/\" HabilitarGuardar:
False Items : 3\"\'");
TestManagerHelpers.AssertAreEqual<string>(this.Manager, " HabilitarGuardar: False Items : 3",
temp5, "return of MsgAckProceesOk_Company_1, state S11");
this.Manager.Comment("reaching state \'S12\'");
string temp6;
this.Manager.Comment("executing step \'call SACM#0.Cancel_Company_1()\'");
temp6 = this.o.Value.Cancel_Company_1();
this.Manager.Comment("reaching state \'S13\'");
this.Manager.Comment("checking step \'return SACM#0.Cancel_Company_1/\" HabilitarGuardar: False
Items : 3"\"\'");
TestManagerHelpers.AssertAreEqual<string>(this.Manager, " HabilitarGuardar: False Items : 3",
temp6, "return of Cancel_Company_1, state S13");
this.Manager.Comment("reaching state \'S14\'");
string temp7;
this.Manager.Comment("executing step \'call SACM#0.Close_1()\'");
temp7 = this.o.Value.Close_1();
this.Manager.Comment("reaching state \'S15\'");
this.Manager.Comment("checking step \'return SACM#0.Close_1/\" HabilitarGuardar: False Items :
3\"\'");
TestManagerHelpers.AssertAreEqual<string>(this.Manager, " HabilitarGuardar: False Items : 3",
temp7, "return of Close_1, state S15");
this.Manager.Comment("reaching state \'S16\'");
this.Manager.EndTest();

}

#endregion
```

El código anterior ejecuta el siguiente escenario: 1. abre la interfaz la interfaz principal, 2. abre el Formulario de Empresa, 3. asigna valores a los campos del Formulario, 4. realiza la acción de guardar la información ingresada en el formulario, 5. envía un mensaje informativo indicando el resultado exitoso del proceso, 6. cierra el formulario, y finalmente cierra la vista principal.

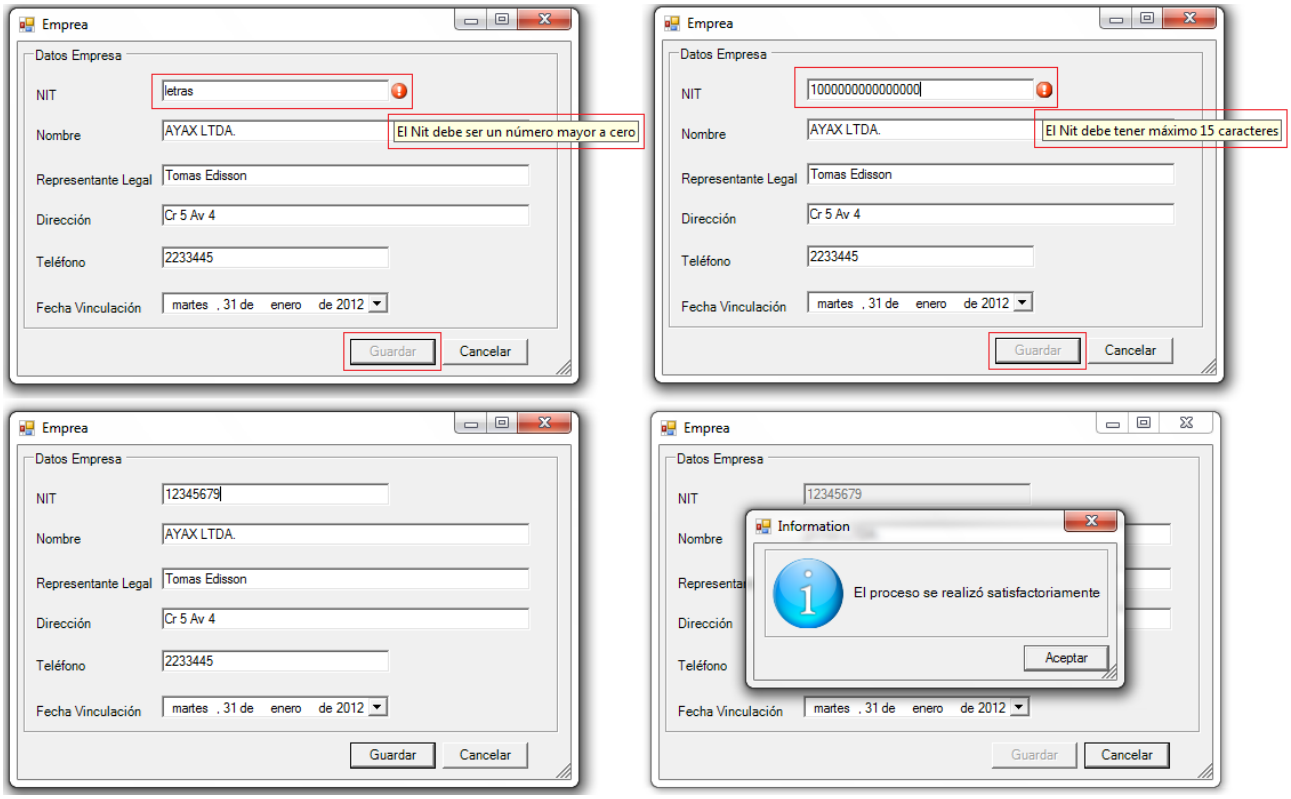
El caso de prueba anterior verifica que la propiedad HabilitarGuardar sea verdadera, luego de ingresar los valores especificados, y el número de ítems resultante sea tres, luego de realizar la acción de guardar. Si la implementación se encuentra totalmente acoplada con el modelo, entonces todas las pruebas pasarán. En la Figura 3-18 se presenta una muestra de 15 casos de pruebas ejecutados sobre el escenario del caso de ejemplo de la Figura 3-1**.** Se puede apreciar que si la respuesta de la implementación es diferente a la esperada, la prueba falla, lo cual indica que el comportamiento de la implementación no está acorde con el comportamiento del modelo.

Figura 3-18: Muestra de la ejecución de 15 casos de pruebas ejecutables para el escenario de creación de empresa utilizando el Formulario de Empresa.



En la Figura 3-19 se presenta el comportamiento de la vista concreta (Formulario de Empresa) luego de que se han pasado las pruebas generadas para el escenario de crear una empresa con el formulario de empresa; el objetivo es verificar que las entradas de usuario sean válidas para poder guardar la información.

Figura 3-19: Prueba de la interfaz gráfica de usuario: *Formulario de Empresa,* luego de acoplar la implementación con el modelo generado.



4. Análisis de Resultados

Se implementó una estrategia de refinamiento iterativo para aplicar MBT sobre las GUIs de los casos de ejemplo, lo cual fue la base para el análisis de resultados. Al final de cada iteración se realizó una ejecución de pruebas manuales, basándose en los requerimientos, para validar los resultados de las pruebas generadas y ejecutadas con el proceso automático. Al término de cada caso de ejemplo desarrollado se obtenían los resultados en cuanto a cobertura y efectividad.

El análisis presentado no pretende generalizar deducciones de aplicar MBT sobre algún sistema, muestra los resultados obtenidos para las GUIs específicas de los casos de ejemplo, con un desarrollo individual del trabajo.

4.1 Análisis de Cobertura

Los criterios de cobertura aplicados en el trabajo se basaron en los soportados por la herramienta (Sección 3.4). Por la manera en que *Spec Explorer* opera, implícitamente los criterios *Interface/Function Criteria* y *All-transitions Coverage* se cubren en su totalidad [26]. El primero se cumple, puesto que todas las reglas definidas en los ScenarioModels deben tener una correspondencia uno a uno con las acciones de los ScenarioImplementations, de lo contrario el proceso de validación interno que realiza la herramienta sobre el modelo no se pasará. El segundo se cumple debido a que internamente la herramienta trasforma todas las transiciones presentes en la FSM en acciones sobre la suite de pruebas que prueban la interfaz.

Los criterios *Statement Coverage Criteria,* y *Boundary Value Coverage*, fueron los medidos finalmente. En la siguiente tabla se consolida los resultados obtenidos para estos dos criterios en los casos de ejemplo desarrollados.

Tabla 4-1: Consolidado de resultados obtenidos para los criterios *Statement Coverage Criteria* y *Boundary Value Coverage.*

Caso de Ejemplo	Escenario	Boundary Value Coverage			Statement Coverage Criteria	
		# Estados	# Transiciones	# Casos de Prueba Generados	Code Coverage Especifico	Code Coverage General
Sumador Simple	Suma	8	38	31	100%	100%
Address Book	FindScenario	21	34	16	~94%	~91%
	SaveScenario	12	12	4	~88%	
Employers Manager	AddNewScenario	804	803	160	~89%	~90%
	DeleteScenario	47	46	12		
	UpdateScenario	487	486	96		
	FindScenario	549	548	108	~88%	

Para el *Boundary Value Coverage* se presenta el número de estados, transiciones y casos de prueba finalmente generados para cada uno de los escenarios que componen los casos de ejemplo (Tabla 4-1). Estos resultados son obtenidos luego de la elección de los valores de prueba de las acciones que componen el modelo y que conducen la generación de la máquina de estados. La herramienta genera un conjunto de casos de prueba a partir de la combinación de todos los valores de prueba de los parámetros especificados en las diferentes acciones. Se seleccionó valores de entrada dentro y fuera de las fronteras para los diferentes casos de ejemplo. Se garantizó que por cada parámetro de entrada se escogiera al menos un valor en la frontera y uno fuera de la frontera, es decir un valor válido y uno inválido por cada regla de validación.

Se pudo notar que el número de estados, transiciones y casos de prueba generados aumenta considerablemente con la complejidad y funcionalidades expuestas por la interfaz. Para los dos primeros casos de ejemplos, que tenían poca funcionalidad, no se generaron más de 50 transiciones; sin embargo para el último caso de ejemplo, que era el más complejo y donde se tenía más dominio del proceso, se generaron más de 1800 transiciones. El número de transiciones, y por ende los casos de prueba generados, está

directamente relacionado a los valores de prueba seleccionados; lo cual impacta el desempeño del proceso de generación de la suite de pruebas de la herramienta.

Para el *Statement Coverage Criteria* se presenta la cobertura de código específico de cada escenario y la cobertura general de toda la implementación para cada caso de ejemplo desarrollado. El primero es la cobertura medida sobre los componentes fundamentales de cada escenario, es decir la vista concreta y el presentador (por ejemplo para el caso del *FindScenario,* del tercer caso de ejemplo, se mide sobre el *PresenterFind* y el *FindDialog*); mientras que el segundo es medido sobre los elementos principales que componen el diseño: *ScenarioImplementations, MainManagerImplementation, MainPresenterImplementation, MainViewImplementation, PresenterImplementations y ConcreteViews*.

El *Code Coverage Especifico* mostró que la suite de pruebas generada realiza una cobertura promedio de ~92% sobre el código del presentador y las vistas concretas para cada escenario; mientras que el *Code Coverage General* es de ~ 94% sobre el código de toda la implementación de la Interfaz Gráfica de Usuario. La cobertura general es mayor a la específica debido a que existen elementos que se prueban el 100% (como los *ScenarioImplementations y el MainManagerImplementation*), haciendo que el promedio de esta métrica se incremente. La cobertura de pruebas es alta (>90%), ya que primero se generaron las pruebas, a partir del modelo, y luego se desarrollaba la implementación para pasar aquellas pruebas.

4.2 Análisis de Efectividad

Efectividad se refiere a la capacidad de lograr el efecto deseado [33], en el contexto de este trabajo el efecto deseado es producir una suite de pruebas con MBT que asegure que la interfaz gráfica de usuario se comporte de acuerdo a los escenarios funcionales especificados, para lo cual todas las acciones, definidas en aquellos escenarios, que se ejecuten sobre la interfaz deben estar libres de defectos o errores.

Para validar que la suite de pruebas generada con el proceso MBT efectivamente está probando la GUI de acuerdo al comportamiento esperado en los requerimientos, al final de cada iteración se realizó una ejecución manual de pruebas (pruebas de usuario final), efectuando las acciones especificadas en los escenarios.

En esta fase de validación podían detectarse defectos (algún tipo de error o resultado no deseado) que no se detectaban con las pruebas generadas con en el proceso automático. Estos errores se presentaban realizando alguna acción específica sobre la interfaz, ya que se ha modelado el comportamiento de la interfaz por un conjunto de escenarios compuestos por un grupo de acciones, se podía asociar el error a una acción específica de un escenario determinado. Por ejemplo, al realizar una prueba manual del escenario de búsqueda (FindCompanyScenario – Anexo D) se presentó un error ejecutando la acción de abrir el dialogo de búsqueda, entonces este defecto se asociaba a la acción FindDialog de este escenario.

Si una acción tenía asociado uno o varios errores, esta acción era denominada acción con defectos. El análisis de efectividad implementado en este trabajo, se enfocó en medir por iteración, cuántas acciones con defectos eran detectadas en la validación manual de usuario y que no eran detectadas por el proceso automático. El número total de acciones es calculado por la suma de todas las acciones independientes que componen los escenarios; así que después de una iteración se podía definir cuántas acciones con defectos existían en relación al total de acciones. No se realizaban más iteraciones, si en las pruebas de validación de usuario final no se detectaban hallazgos (defectos); las pruebas manuales estaban dirigidas a probar los escenarios con casos típicos y en cada iteración se realizaban las mimas pruebas.

Las variables que se midieron para cada iteración de cada caso de ejemplo fueron:

- ✓ *Total Acciones*: Son las acciones que se pueden realizar sobre la interfaz, de acuerdo a los escenarios especificados. Se suman las acciones independientes que componen los escenarios, ya que algunas acciones pueden pertenecer a uno o más escenarios.
- ✓ *Total Defectos:* Es el total de defectos (errores o resultados no esperados) obtenidos luego de la validación de usuario final, realizada en cada iteración.
- ✓ *Acciones con Defectos:* Si se detectaba uno o más defectos asociados a una acción, esta se marcaba como una acción con defectos. Al final de cada iteración se sumaban todas las acciones con defectos detectadas.
- ✓ *% Acciones con Defectos:* Relación entre las acciones con defectos respecto al total de acciones. (*Acciones con Defectos / Total Acciones * 100*).

- ✓ *% Efectividad Específica:* Para el contexto de este trabajo se calculó de la siguiente manera: *100 - % Acciones con Defectos.* Refleja la efectividad por cada iteración con el enfoque planteado.
- ✓ *% Efectividad General.* Promedio de efectividad para cada caso de ejemplo. Se obtiene promediando el porcentaje de efectividad específica de las iteraciones realizadas. Refleja el desempeño general obtenido para cada caso de ejemplo.

En la Tabla 4-2 se presenta el consolidado de resultados para las métricas anteriores. En la Tabla 4-3 se presenta el desglose detallado por cada caso de ejemplo.

Tabla 4-2: Consolidados de resultados para medir la efectividad de aplicar MBT en los caso de ejemplo.

Caso Ejemplo	Total Acciones	Iteración	Total Defectos	Acciones con Defectos	% Acciones con Defectos	% Efectividad Específica	% Efectividad General
Sumador Simple	2	1	6	2	100,00	0,00	50,00
		2	0	0	0,00	100,00	
Address Book	8	1	12	5	62,50	37,50	68,75
		2	7	3	37,50	62,50	
		3	4	2	25,00	75,00	
		4	0	0	0,00	100,00	
Employers Manager	16	1	10	7	43,75	56,25	79,17
		2	3	3	18,75	81,25	
		3	0	0	0,00	100,00	

Tabla 4-3: Detalle de resultados para cada caso de ejemplo desarrollado.

a) Resultados para el caso de ejemplo: Sumador Simple.

Caso Ejemplo	Acción	Iteración 1		Iteración 1	
		Defectos	Acción con defecto	Defectos	Acción con defecto
Sumador Simple	Suma	4	SI	0	NO
	ReadAnd Reset	2	SI	0	NO
Total Acciones		2			
Total Defectos		6		0	
Acciones con Defectos		2		0	
% Acciones con Defectos		100		0	
% Efectividad Específica		0		100	
% Efectividad General		50			

b) Resultados para el caso de ejemplo: Address Book.

Caso Ejemplo	Acción	Iteración 1		Iteración 2		Iteración 3		Iteración 4	
		Defectos	Acción con defecto	Defectos	Acción con defecto	Defectos	Acción con defecto	Defectos	Acción con defecto
Address Book	LaunchAddressBook	0	NO	0	NO	0	NO	0	NO
	Cancel	0	NO	0	NO	0	NO	0	NO
	SelContact	0	NO	0	NO	0	NO	0	NO
	FindViewScenario	4	SI	3	SI	2	SI	0	NO
	SetDirty	1	SI	0	NO	0	NO	0	NO
	SetFileOpened	1	SI	0	NO	0	NO	0	NO
	SaveViewScenario	4	SI	3	SI	2	SI	0	NO
	MsgOverwriteFile	2	SI	1	SI	0	NO	0	NO
Total Acciones		8							
Total Defectos		12		7		4		0	
Acciones con Defectos		5		3		2		0	
% Acciones con Defectos		62,5		37,5		25		0	
% Efectividad Específica		37,5		62,5		75		100	
% Efectividad General		68,75							

c) Resultados para el caso de ejemplo: Employers Manager.

Caso Ejemplo	Acción	Iteración 1		Iteración 2		Iteración 3	
		Defectos	Acción con defecto	Defectos	Acción con defecto	Defectos	Acción con defecto
Employers Manger	LaunchView_VP	0	NO	0	NO	0	NO
	SetSelectedItem_VP	0	NO	0	NO	0	NO
	Close_VP	0	NO	0	NO	0	NO
	AddNewCompany_N	0	NO	0	NO	0	NO
	CompanyViewScenario_N	2	SI	0	NO	0	NO
	Guardar_N	2	SI	1	SI	0	NO
	Cancel_N	1	SI	0	NO	0	NO
	UpdateCompany_U	0	NO	0	NO	0	NO
	CompanyViewScenario_U	1	SI	0	NO	0	NO
	Guardar_U	2	SI	1	SI	0	NO
	Cancel_U	0	NO	0	NO	0	NO
	DeleteCompany_D	0	NO	0	NO	0	NO
	FindDialog_F	0	NO	0	NO	0	NO
	FindScenario_F	1	SI	0	NO	0	NO
	Find_F	1	SI	1	SI	0	NO
	Cancel_F	0	NO	0	NO	0	NO
Total Acciones		16					
Total Defectos		10		3		0	
Acciones con Defectos		7		3		0	
% Acciones con Defectos		43,75		18,75		0,00	
% Efectividad Específica		56,25		81,25		100,00	
%Efectividad General		**79,17**					

En la iteración final de cada caso de ejemplo se lograba conseguir una efectividad del 100% (que no hubiera errores en las acciones que se podían realizar sobre la GUI objetivo), lo que reflejaba que la suite de pruebas generada con MBT, realmente prueba el comportamiento deseado de la interfaz, de acuerdo a los requerimientos y escenarios especificados. Sin embargo se evaluó la efectividad general del proceso aplicado en cada ejemplo, que es el promedio del valor obtenido en las iteraciones. Esta métrica está directamente relacionada a la complejidad de la interfaz (lo que se refleja en la cantidad

de acciones) y al número de iteraciones que hubo que realizar para no detectar errores. Como se puede apreciar, a pesar de que aumenta la complejidad en cada caso de ejemplo, la efectividad general aumentó, ya que con cada caso de ejemplo ejecutado se iba madurando la manera de aplicar MBT sobre GUIs con el enfoque planteado. Para el último caso de ejemplo se obtuvo una efectividad general de ~80%; aunque era el caso con más complejo y con más funcionalidades, se logró depurar el proceso en tres iteraciones.

También se analizó las posibles causas de los defectos encontrados, ya que éstos podrían ser el resultado de una especificación inadecuada de los escenarios, de un modelado deficiente o de una mala implementación. En la Tabla 4-4 se despliega el consolidado de las causas de los defectos detectados para cada iteración de cada caso de ejemplo. Los errores obtenidos se distribuían aproximadamente en un 50% en el modelado, un 30% en la especificación de los escenarios y un 20% en la implementación. Los errores presentes en el modelo se debían a falta de detalle de las precondiciones y poscondiciones de las acciones definidas o un mal manejo de la herramienta para configurar la exploración del mismo.

El aspecto más crítico del proceso de MBT es el diseño y construcción del modelo, si éste no estaba bien especificado, finalmente las pruebas generadas no probaban adecuadamente el comportamiento deseado de la GUI y por ende la implementación también quedará defectuosa; sin embargo si el diseño del modelo esta refinado, al hacer que el SUT pase todas las pruebas generadas, se está comprobando que el comportamiento es el mismo al esperado por los requerimientos. En la última iteración de cada caso de ejemplo se lograba obtener el comportamiento deseado, para los ejemplos desarrollados no se superaron las 4 iteraciones.

Tabla 4-4: Causa de los defectos detectados para cada iteración de cada caso de ejemplo.

Caso Ejemplo	Iteración	Total Defectos	Fuente del Defecto	Cantidad
Sumador Simple	1	6	Esp .Escenarios	1
			Modelado	3
			Implementación	2
	2	0	Esp .Escenarios	0
			Modelado	0
			Implementación	0
Address Book	1	12	Esp .Escenarios	3
			Modelado	6
			Implementación	3
	2	7	Esp .Escenarios	2
			Modelado	3
			Implementación	2
	3	4	Esp .Escenarios	0
			Modelado	2
			Implementación	2
	4	0	Esp .Escenarios	0
			Modelado	0
			Implementación	0
Employers Manager	1	10	Esp .Escenarios	2
			Modelado	3
			Implementación	6
	2	4	Esp .Escenarios	0
			Modelado	2
			Implementación	2
	3	0	Esp .Escenarios	0
			Modelado	0
			Implementación	0

En la Tabla 4-5 se presenta el consolidado de esfuerzo entre el desarrollo de la implementación y el desarrollo del modelo para los diferentes casos de ejemplo. En los dos primeros ejemplos, en los cuales se estaba comprendiendo y consolidando los conocimientos del proceso MBT y el manejo de la herramienta, se tomó más tiempo desarrollando el modelo que desarrollando la implementación. Sin embargo en el último

caso de ejemplo, donde se tenía mayor dominio, el desarrollo del modelo tomó menos tiempo que el desarrollo de la implementación.

Tabla 4-5: Relación de esfuerzo entre la implementación y el desarrollo.

Caso de Ejemplo	Relación Tiempo Implementación /Modelado
Sumador Simple	0.42
Address Book	0.6
Employers Manager	1,5

4.3 Ejecución de los Casos de Prueba

Las pruebas se enfocan en probar el correcto comportamiento del presentador de la GUI, ya que éste gestiona, actualiza y manipula la interfaz gráfica de usuario (la Vista) e implementa la lógica de presentación. En fase de pruebas el presentador emula las acciones de usuario sobre la interfaz gráfica, por ejemplo realiza acciones para disparar eventos de Clicks sobre botones, de actualización de datos, de ingreso de datos, de selección de datos, etc., y es quien le dice a la interfaz cómo actuar y responder ante la presencia de esos eventos.

Al realizar una ejecución de la suite de pruebas se abre la interfaz gráfica de usuario principal y se le aplican, de manera automática, las acciones que realizaría un usuario de acuerdo a los escenarios de pruebas. Al final del proceso se pudo generar una gran cantidad de casos de prueba (en el último caso de ejemplo 376) sobre una interfaz gráfica de usuario y ejecutarlos, en promedio, en menos de dos minutos[2], algo que manualmente nunca se podría lograr.

Si se desea ejecutar la misma suite de pruebas pero sobre otra interfaz gráfica de usuario de otra tecnología, no es necesario volver a realizar el todo el proceso de

[2] La ejecución automatizada de las pruebas fue realizada en un PC de las siguientes características generales: procesador Intel® Core(TM) i5 de 2.40 GHz, 4 GB de RAM y de sistema operativo Windows 7 de 32 bits.

modelado y generación de pruebas, sino simplemente concretizar los detalles de implementación de la nueva interfaz gráfica de usuario y ejecutar las pruebas, si estas fallan se puede ir refinando la nueva interfaz progresivamente. Esto debido a que con la implementación de patrón MVP, sugerido en este trabajo para poder implementar MBT sobre GUIs, la comunicación no es directa entre el presentador y la vista, sino que es a través de una interface que desacopla dichos elementos; por lo cual el presentador no conoce de los detalles específicos de implementación de la vista (GUI), así que al cambiar de tecnología de la GUI, bastaría solo con implementar las funcionalidades expuestas por la interface y esto sería totalmente transparente para el presentador, ya que este continuaría comportándose de la misma manera y gestionando la nueva vista por medio de la interface [25].

En cada iteración, luego de generar la suite de pruebas con MBT, se desarrollaba gradualmente la GUI concreta hasta que se pasaran todas las pruebas; este proceso de primero realizar las pruebas y posteriormente ir desarrollando y refinando poco a poco la implementación hasta que las pruebas pasen, puede ser aplicado en un proceso de desarrollo de software iterativo e incremental guiado por pruebas.

5. Conclusiones

5.1 Aportes

Se planteó como estrategia de implementación de MBT para automatizar la generación y ejecución de casos de pruebas sobre GUIs, una metodología iterativa, en donde progresivamente se va refinando cada una de sus fases hasta alcanzar el comportamiento deseado de la GUI. Se describe desde la selección de aspectos a modelar hasta la manera de modelar y diseñar la GUI, lo cual podría servir como base preliminar para la automatización de pruebas sobre estos componentes en un proceso de desarrollo de software.

Se propuso dividir el modelado del comportamiento de una GUI en dos fases fundamentales: en primera instancia se especifica el modelo de pruebas de la interfaz, definiendo pre y pos condiciones para cada una de las acciones definidas y posteriormente, configurando diferentes escenarios y los valores de prueba de los parámetros de las acciones, se construye una máquina de estados finita que visualiza los estados y transiciones del comportamiento deseado de la GUI ante la interacción de usuario.

Los aspectos propuestos para modelar una GUI, son aquellos que están en términos de la lógica de presentación y están descritos con base en los requerimientos y casos de prueba iniciales. Estos aspectos podrían ser aplicados en cualquier GUI y pueden agruparse en: propiedades que modelan el estado de la interfaz, propiedades que modelan la habilitación de controles, propiedades que modelan las ventanas activas, acciones que modelan el despliegue apropiado de mensajes a usuario, acciones que modelan el manejo de la interfaz y acciones que modelan los escenarios base de las funcionalidades. En el desarrollo de los casos de ejemplo se presentan ejemplos concretos de las propiedades modeladas.

Para llegar a la automatización de pruebas sobre GUIs aplicando MBT, se propuso diseñar la GUI adoptando el patrón MVP. El comportamiento de la vista se prueba implícitamente al probar el correcto funcionamiento del presentador, ya que al fin de cuentas éste es el encargado de gestionar la actualización, respuesta y validación final de la GUI. Con este enfoque no es necesario saber los detalles específicos de implementación de la vista, pues se establece un contrato que desacopla la comunicación entre presentador y ésta; por ello, a diferencia de los demás trabajos realizados en el campo, no será necesario conocer la tecnología de la GUI ni construir previamente un prototipo poder aplicar el método planteado.

Se planteó desarrollar la GUI concreta luego de la generación automatizada de los casos de pruebas con MBT, es decir siguiendo los lineamientos de un desarrollo de software dirigido por pruebas. Con esto se garantiza que solo se escriba código útil que busque pasar aquellas pruebas; razón por la cual con el método propuesto se podrá obtener una cobertura de código alta (mayor al 90%).

5.2 Resultados

Los errores detectados en los casos de ejemplo se distribuyeron de la siguiente manera: 20% en la implementación, 50% en el modelado y 30% en la especificación de los escenarios. La causa principal de este comportamiento fue una aplicación inadecuada del proceso MBT en instancias iniciales, pues se percibía, en la etapa de validación, que el modelo generado realmente no abstraía adecuadamente los requerimientos y no especificaba el comportamiento esperado de la GUI; como el modelo es la parte más crítica y fundamental del proceso, si este no estaba bien definido las demás etapas fallaban.

Con el método implementado de MBT, se logró obtener una cobertura de código promedio mayor al 90%, lo que garantiza que se está probando la mayor cantidad de código de la implementación. El 10% restante no era probado debido a código generado por la herramienta de diseño de la GUI para capturar los eventos reales desencadenados por el usuario; en fase de pruebas se simulaba la ejecución de estos eventos. Esta métrica era medida por el *Code Coverage Result* de la herramienta para el criterio de cobertura *Statement Coverage Criteria.*

La métrica empleada para calcular la efectividad se basó en medir la cantidad de acciones con defectos en la fase de validación manual y que no eran detectadas con las pruebas generadas y ejecutadas con el proceso automatizado. Se logró pasar de una efectividad general de ~50%, en el primer caso de ejemplo, a una efectividad de ~80%, para el último caso de ejemplo. Esto debido a que en el último caso, a pesar de ser el más complejo, fue donde se tenía más domino de la aplicación del proceso MBT y de la implementación de la arquitectura base.

La métrica empleada para calcular la efectividad general está directamente relacionada al número de acciones que se pueden ejecutar sobre la GUI y al número de iteraciones que hubo que realizar para no detectar errores. En la última iteración, luego de una validación de usuario final, siempre se lograba no detectar acciones con defectos (100% de efectividad específica). Sin embargo si se realizaban muchas iteraciones la efectividad general disminuía, además si el número de acciones con errores detectados, comparado con el total de acciones, era alto, también la efectividad general disminuía.

Para llegar a tener una efectividad general del 100% en la implementación de MBT sobre GUIs, es decir que no se detecten errores en una validación de usuario final, se tendría que garantizar que el modelo refleje totalmente el comportamiento deseado en una sola iteración. Esto puede llegar a ser complejo y quizás a pesar del esfuerzo inicial, puedan detectarse errores al final. Una manera práctica podría ser realizar el proceso iterativo (tal como se implementó en el desarrollo de este trabajo) e ir refinando progresivamente el modelo hasta obtener el comportamiento deseado.

En el desarrollo de los casos de ejemplo, la construcción del modelo, a partir del cual se generaron los casos de prueba, en promedio implicó más tiempo y esfuerzo que el desarrollo de la implementación; sin embargo finalmente se logró obtener el comportamiento deseado de la GUI, de acuerdo a los requerimientos de los casos de uso, y generar una gran cantidad de casos de pruebas que abarcan, con una buena cobertura de ejecución de pruebas (> 90%), todas las funcionalidades expuestas por la GUI.

5.3 Lecciones Aprendidas

La mayoría de los problemas detectados en la especificación de escenarios se debían a que no estaban bien detalladas las acciones que abstraían las funcionalidades de la GUI. Por lo general luego de cada iteración se debían refinar las acciones existentes o adicionar nuevas acciones, para complementar los escenarios.

Para obtener una mejor compresión de los requerimientos, una buena estrategia es documentar y detallar escenarios típicos y escenarios alternativos, y para cada uno de ellos especificar casos de pruebas exitosos y no éxitos. Al tener este detalle formal se hace más comprensible el diseño y construcción del modelo.

Con el criterio de cobertura seleccionado se deben escoger los valores límites o de frontera y los que podrían llegar a ocasionar un error. Se recomienda escoger un valor por cada frontera (valores límites válidos) y un valor fuera de cada frontera (valores límites inválidos), ya que la herramienta realiza una generación con la combinación de todos los valores especificados; lo que involucra que a mayor cantidad de valores, mayor será el tiempo y recursos de procesamiento necesarios.

Los errores en el modelado se presentaron por dos razones principalmente: falta de detalle en la especificación de las precondiciones y poscondiciones, y un manejo inadecuado de la herramienta en la configuración del modelo. Con la práctica y experiencia de la aplicación de MBT empleando Spec Explorer, se podría obtener la madurez suficiente para ser más certero en la etapa de modelado de la GUI y poder conseguir los resultados esperados en pocas iteraciones.

Es necesario modelar los eventos de respuesta de la vista, además de las acciones que el usuario realiza sobre ésta. En la herramienta estos eventos se deben especificar como acciones observables en los *ScenarioModels.* Se debe tener en cuenta que estas acciones observables siempre se ejecuten en un orden específico, de lo contrario habrá errores de estados inválidos en el proceso de validación y exploración de la FSM efectuado por la herramienta.

Es necesario agrupar las variables que representan el estado principal de la GUI en una propiedad de tipo de dato primitivo (como *strings, enteros, booleanos*, etc.), ya que la

FSM generada por la herramienta no soporta la visualización de tipos de datos complejos, razón por la cual se recomienda que el valor de dicha propiedad sea un texto que represente el consolidado de las variables.

Al aplicar el patrón MVP, se debe tener en cuenta que en la interface (contrato), definida para comunicar la vista y el presentador, se incluyan, además de las propiedades y operaciones que representan las funcionalidades propias de la GUI, los métodos que disparan los eventos desde la vista y que emulan las acciones ejecutadas por el usuario. Cuando no se especifican ni se implementan estas operaciones no se logra el efecto de la automatización de la ejecución de las pruebas sobre la GUI.

Al aplicar el patrón MVP en vistas compuestas es necesario validar que el presentador principal (el cual gestiona el resto de presentadores) se suscriba a los eventos de las vistas hijas para poder actualizar y controlar adecuadamente la vista principal de acuerdo a los estados de las ventanas, diálogos o controles contenidos.

5.4 Restricciones y Recomendaciones

Para poder implementar MBT con el método propuesto, la capa de presentación de la aplicación se debe diseñar bajo los lineamientos del patrón MVP, ya que las pruebas generadas apuntan a probar el correcto funcionamiento del presentador; si este componente no existe, no será posible probar la GUI. En fase de pruebas se emulan los eventos disparados por el usuario y el presentador es quien responde a estos eventos y le indica la vista cómo comportarse. La comunicación entre la vista y el presentador no es directa si no es a través de una interface que desacopla estos dos elementos.

La estrategia está orientada a ser aplicada al inicio de un proyecto de desarrollo de software en donde se tenga en cuenta el patrón MVP para el diseño de los diferentes componentes que integran la capa de presentación. No se recomienda aplicarla a interfaces gráficas de usuario de aplicaciones ya desarrolladas, pues sería necesario hacer un gran esfuerzo para migrar la arquitectura o realizar una ingeniería inversa para re-diseñar esta capa de presentación.

El enfoque eta orientado ser aplicado a interfaces graficas de usuario, como formularios y vistas compuestas, de uso común en aplicaciones de escritorio, es decir que se ejecuten directamente en la máquina local aprovechando los recursos de hardware del equipo.

La aplicación del proceso de pruebas basadas en modelos con el enfoque propuesto implica tiempo y esfuerzo inicial, ya que es totalmente dependiente de la experticia que se tenga en el manejo de MBT y en el domino de la arquitectura planteada; por lo tanto el método se recomienda aplicarlo con programadores sénior y arquitectos de software con buenas capacidades de análisis y abstracción.

Para poder aplicar la metodología presentada en un proceso formal de desarrollo de software empresarial se recomienda implementar la trazabilidad de requerimientos y bugs, que es una de las características principales que brinda MBT para llevar el seguimiento automatizado de la evolución y cambios de la aplicación. Para esto sería necesario buscar una herramienta comercial de MBT que implemente de manera robusta el manejo de la trazabilidad en la configuración y diseño del modelo (para una revisión detallada referirse a [31, 32]).

5.5 Trabajo Futuro

Se utilizó una herramienta que permite cubrir el ciclo completo de las pruebas basadas en modelos: Spec Explorer 2010 de Microsoft. Sin embargo no se exploró la manera de implementar la trazabilidad de requerimientos. Un paso siguiente, sería implementar la trazabilidad de requerimientos e integrarse con alguna herramienta robusta de modelado gráfico (como extensiones UML), para optimizar la fase de especificación de los escenarios.

El trabajo estaba orientado a probar la lógica de presentación de una GUI, el siguiente paso sería extender la metodología para probar la lógica del negocio. Integrando las pruebas de la lógica de presentación y la del negocio se puede llegar a realizar un nivel de pruebas de sistema mucho más robusto.

Integrarse formalmente a una metodología de desarrollo orientado en pruebas como TDD (Test Driven Development) o ATDD (Acceptance Test Driven Development). Como con MBT es posible primero generar las pruebas, en este caso el modelo a partir del cual se

generen la suite de pruebas, y luego desarrollar la implementación que pase aquellas pruebas, es decir quede acoplado con el comportamiento definido en el modelo; se puede ajustar a un desarrollo de software orientado a pruebas (para mayor detalle referirse a [1]).

Extender la implementación propuesta para probar interfaces gráficas de entornos web; para esto sería necesario hallar una herramienta MBT que generen pruebas en código ejecutable para lenguajes que interactúen con documentos dinámicos HTML, HTML-5 y XML, como por ejemplo JavaScritp o Jquery.

Los resultados presentados están orientados al desarrollo individual del trabajo, es decir el autor era quien abstraía los requerimientos, construía el modelo y realizaba la validación final en cada iteración. Habría que aplicar la estrategia, presentada en el trabajo para implementar MBT sobre GUIs, en un equipo de desarrollo compuesto por personas distintas para poder llegar a generalizar los resultados y comparar los mismos con los obtenidos.

A. Anexo: Spec Explorer 2010

Spec Explorer [15] es una herramienta de Microsoft diseñada para aplicar el proceso de Pruebas Basadas en Modelos (*Model Based Testing,* MBT), en el cual se puede modelar el comportamiento del software, analizar aquel comportamiento por medio de una visualización gráfica, validar el modelo creado y generar pruebas ejecutables a partir de éste para poder probar la implementación.

Spec Explorer emplea modelos de pre/post condiciones que son escritos en una versión extendida de c#, denominado Spec#. Las siguientes son las principales características de este lenguaje [1]:

- ✓ Contiene un sistema de tipos más robusto, en el cual para cualquier tipo de objeto existe su correspondiente tipo no nulo.
- ✓ Se pueden especificar métodos empleando pre-condiciones (requisitos antes de ejecutar el método), post-condiciones (validaciones que deben cumplirse luego de ejecutarse el método), condiciones marco (atributos) y especificación de excepciones.
- ✓ Amplio soporte al manejo de objetos invariantes, con lo cual se puede tener más control cuando se actualiza un objeto.
- ✓ Soporte de cuantificadores y comprensión de expresiones para estructura de datos, con esto es más sencillo especificar propiedades complejas de aquellas estructuras.

El comportamiento puede ser modelado de dos formas: definiendo directamente las reglas (estableciendo el estado de los datos dinámicamente) o definiendo escenarios como patrones de acciones. Con estas técnicas se permite al usuario generar un conjunto de pruebas relevantes, pues se limitan los escenarios de una máquina de estado densa, es decir se controla el problema de explosión de estados del modelo.

A.1 Modelos

Como Spec Explorer es inherentemente una herramienta para realizar el proceso de pruebas basadas en modelos, entonces permite generar los modelos del software bajo prueba (System Under Test, SUT). En general, aquellos modelos poseen las siguientes características principales:

- ✓ Los modelos representan una abstracción de un sistema desde una perspectiva en particular.
- ✓ Los modelos soportan la exploración, construcción y predicción del comportamiento del sistema modelado.

Con la herramienta se pueden crear modelos de comportamiento de un sistema. Un modelo de comportamiento describe cómo el sistema de software reacciona ante una serie de entradas en particular; la reacción es definida por las salidas que el sistema produce (estado resultante), un ejemplo de estos modelos son las máquinas de estado finito.

A.2 Acciones

En general, una acción describe una interacción con el SUT, visto desde la perspectiva del sistema de pruebas. La aplicación reconoce tres clases de acciones atómicas que son discretas e indivisibles: Acciones de Llamada (*Call Actions*), Acciones de Retorno (*Return Actions*) y Eventos (*Events*).

Una acción de llamada representa un estímulo del sistema de pruebas a el SUT, una acción de retorno es la respuesta a aquel estímulo además que captura la respuesta, si la hay, del SUT. Los eventos por su parte, representan mensajes autónomos generados desde el SUT.

Todas las acciones deben ser declaradas en el archivo de configuración (que maneja una estructura denominada ***Cord***). Tales declaraciones asumen que existe una correspondencia en los nombres para poder acceder a la implementación del SUT. Existen dos tipos de declaraciones de Acciones:

- ✓ *Acción call/return*. Estas declaraciones son un par de acciones atómicas, que representan operaciones sincrónicas en el sistema que está siendo modelado. En la práctica esto significa que el sistema realizará la operación antes de retornar el control al cliente (objeto que invocó la llamada), es decir una vez el cliente invoca la acción, no realizará alguna otra operación o esperará alguna otra respuesta, hasta que la operación sea completada. En el SUT objetivo estas acciones son implementadas como métodos normales.
- ✓ *Acción event.* Esta clase de declaraciones son acciones atómicas y representan una respuesta generada desde el sistema que está siendo modelado. La respuesta puede ser autónoma y corresponder a una solicitud recibida previamente. Los eventos son usados cuando el sistema puede emitir respuestas asincrónicas, es decir el sistema puede enviar solicitudes que no son inmediatamente procesadas y ejecutadas, sino que entran a una cola de espera. Los eventos de esta cola de espera son consumidos cuando la prueba alcanza un punto en el cual uno o más eventos son esperados.

A continuación se presenta un fragmento indicando cómo se declaran estas acciones:

```
// Código Cord
config AConfig
{
   // Acción call-return:
   action int Implementation.AnAction(string a);
   //Acción event:
   action event void Implementation.AnEventAction(string a);
}
```

A.3 Reglas

En Spec Explorer, las reglas son representadas por métodos en el programa del modelo que han sido marcados con el atributo ***RuleAttribute***. Este atributo puede obtener un argumento opcional denominado ***Action*** que especifica una operación de invocación asociada con la regla.

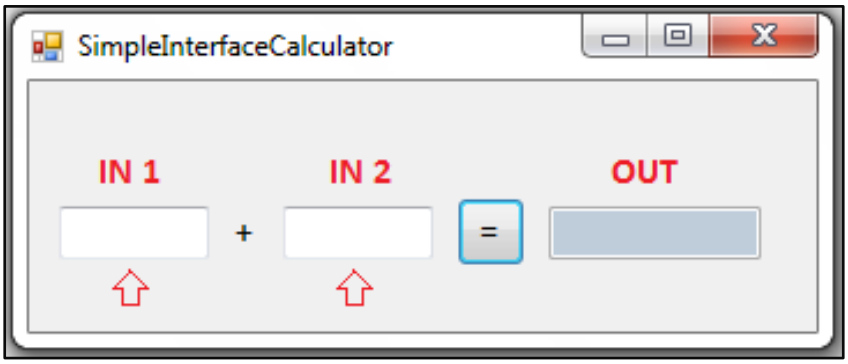
A continuación se muestra un ejemplo en Cord, y su correspondiente invocación en lenguaje c#, desde el programa del modelo.

```
// En Cord
action abstract Sum(int x);
// C# en el Programa del Modelo
static int stateTotal;
[Rule(Action = "this.Sum(op) / result")]
int AddtoTotal(int op)
{
   return stateTotal += op;
}
```

B. Anexo: Sumador Simple

En este caso de ejemplo el objetivo es modelar y probar una interfaz gráfica simple, en este caso implementaremos un sumador sencillo para números enteros mayores o iguales a cero (Figura B-1). Tomaremos el enfoque de verificar las entradas ingresadas por el usuario para que el proceso de suma sea válido y los resultados sean los esperados de la operación [5],[8].

Figura B-1: Interfaz Sumador Simple objeto del caso de ejemplo para la aplicación del proceso de pruebas basadas en modelos.



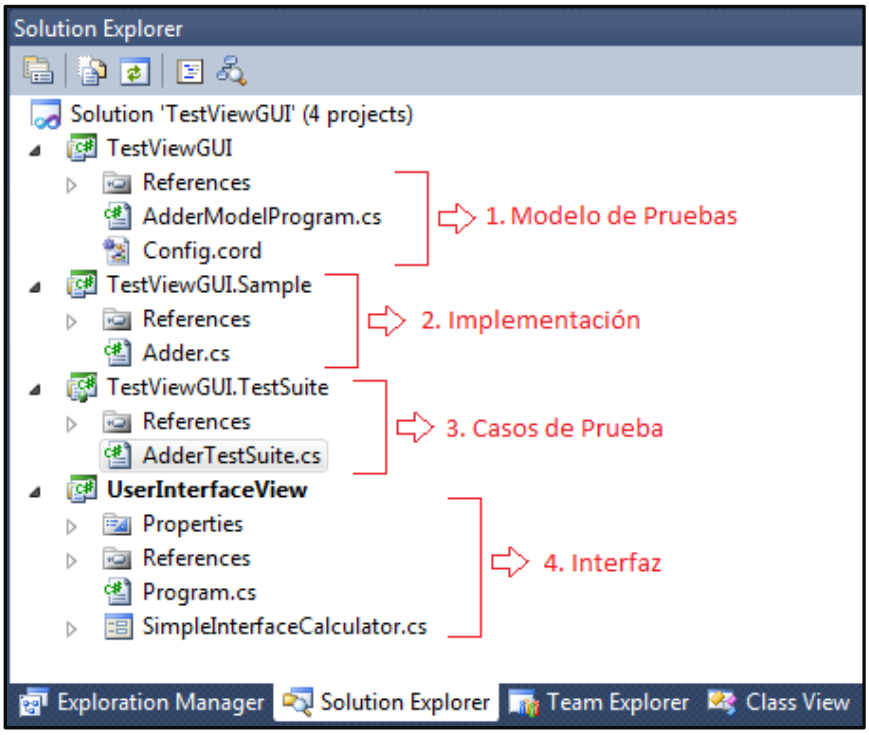
Se implementarán las siguientes reglas:

- ✓ El texto ingresado en IN1 y en IN2 pueda ser mapeado a un número.
- ✓ $IN1 \geq 0 \rightarrow IN1 \in Z$
- ✓ $IN2 \geq 0 \rightarrow IN2 \in Z$
- ✓ $OUT \geq 0 \rightarrow OUT \in Z$

B.1 Ambiente

Describiremos brevemente el ambiente de la solución generado desde la herramienta para tratar el caso de ejemplo. En la siguiente figura puede ser visualizada la estructura de trabajo.

Figura B-2: Estructura de la solución generada para el caso de ejemplo.



A continuación se presenta una descripción breve de los proyectos que participan en la solución.

- ✓ Proyecto TestViewGUI: Este contiene el modelo, el cual esta descrito en dos archivos básicos; en el archivo AdderModelProgram se programa como tal el modelo (se implementan las acciones y reglas) y en el archivo Config.cord se configura el comportamiento y control del modelo, como por ejemplo las entradas, la forma de exploración, la forma de generar los esquemas de transición, etc.
- ✓ Proyecto TestViewGUI.Sample: Contiene la implementación bajo prueba (SUT). Este proyecto como tal puede contener directamente la implementación de código o un adaptador que conecta con el código de implementación.

- ✓ Proyecto TestViewGUI.TestSuite: Provee la suite de pruebas de la forma de pruebas de unidad de VSTT (Visual Studio Team Test). Este archivo será sobre-escrito por el proceso de generación de pruebas.
- ✓ Proyecto UserInterfaceView. Es donde está la implementada la interfaz concreat de usuario. Ésta es invocada desde el proyecto TestViewGUI.Sample.

B.2 Modelado

Para modelar la interfaz vamos a definir su comportamiento como la combinación de dos variables, la primera define si el estado de la interfaz es Válida (*IsValid*), dependiendo si las entradas de texto cumplen con las condiciones necesarias para realizar el proceso de suma, la segunda es como tal el Resultado de la suma (*Suma*).

Para esto se establecieron dos reglas que describen las condiciones para la transición entre estados. La regla *AdderRule* verifica que efectivamente las entradas sean números enteros mayores o iguales a cero, si esta condición se cumple el estado será válido y la suma puede ser efectuada, de lo contrario el estado será invalido y el resultado de la suma será vacío. La regla *ReadAndResetRule* modela la acción de leer el valor actual de la suma y regresar el valor de la suma a su estado inicial, es decir vacío. El siguiente segmento de código muestra el cascaron de la especificación del modelo.

```
Variable que indica si la interfaz el válida
 static bool _IsValid;
 Variable que almacena el estado de la Suma
static string _Suma;
 Regla que valida que los textos ingresados sean numéricos.
 [Rule(Action = "Suma(x,y)/result")]
static bool AdderRule(string x, string y)
{
}
 Regla que modela la acción de resetear el resultado de la Suma.
 [Rule(Action = "ReadAndReset()/result")]
 static string ReadAndResetRule()
 {
}
```

B.3 Configuración

El archivo de configuración del modelo (Cord) especifica cómo el modelo será transformado a una máquina de estados finita, lo cual habilita la posibilidad de exploración. En nuestro caso de ejemplo podemos notar que el modelo tiene un dominio y espacio de estados infinito, pues la entradas de texto y el resultado podrían tomar cualquier valor; esto puede limitarse en Spec Explorer para que la exploración del modelo sea manejable y coherente.

Las máquinas que son definidas para el modelo son contenidas en el archivo *Config.cord* como sigue:

```
Contiene la configuración, acciones y límites del modelo.
config Main
{
        Las dos acciones que serán modeladas, implementadas y probadas.
         action abstract static bool Adder.Suma(string x, string y);
          action abstract static string Adder.ReadAndReset();
}
Límites de validación de parámetros para la Suma
config ParameterCombination: Main
{
          action abstract static bool Adder.Suma(string x, string y)    where
                  {.    Condition.In(x, "a","15");
                        Condition.In(y,"25","-1");      };
}
Se especifica una máquina para el modelo. Como el modelo es no finito, la
exploración se realiza teniendo en cuenta los límites configurados anteriormente.

machine AdderModelProgram()

{

       construct model program from ParameterCombination

}
```

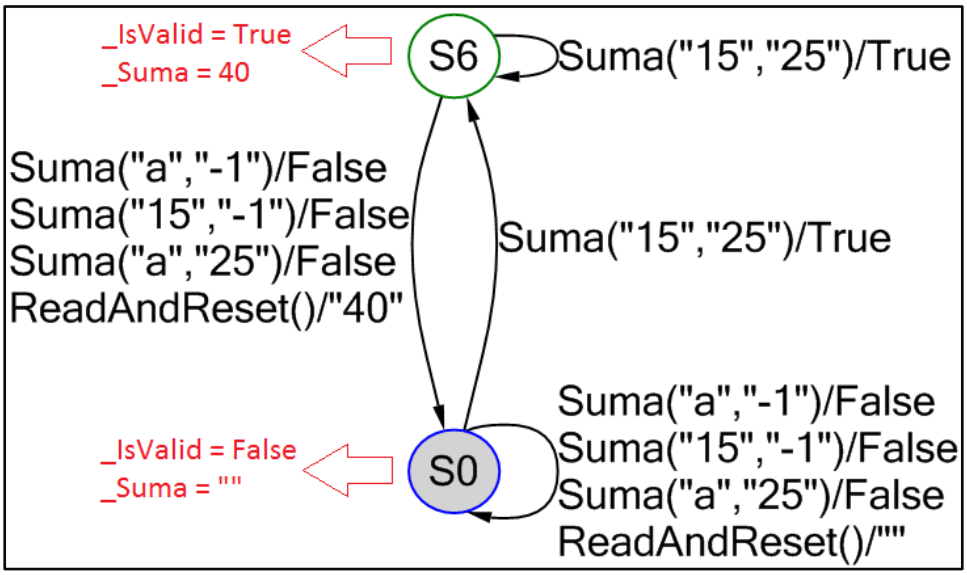
Como podemos observar en el segmento de código anterior, hemos limitado el dominio de las entradas de la suma para poder tener un espacio finito de pruebas. En este caso hemos configurado dos parámetros de entrada inválidos (*a y -1*) y dos válidos (*15 y 25*), con lo cual se generarán los casos de prueba con la combinación de estos conjuntos.

B.4 Exploración del Modelo

En la Figura B-3 se muestra la máquina de estados resultante para el modelo especificado, esta es construida por Spec Explorer basándose en el lenguaje propietario ASML (*Abstract State Machine Language [16]*). Como podemos observar existen dos estados en el modelo, dados por las variables *_IsValid* y *_Suma*, S6 y S0. El primer estado indica que la interfaz es válida y es posible realizar la suma, el segundo estado indica que la interfaz no es válida y no es posible realizar la operación.

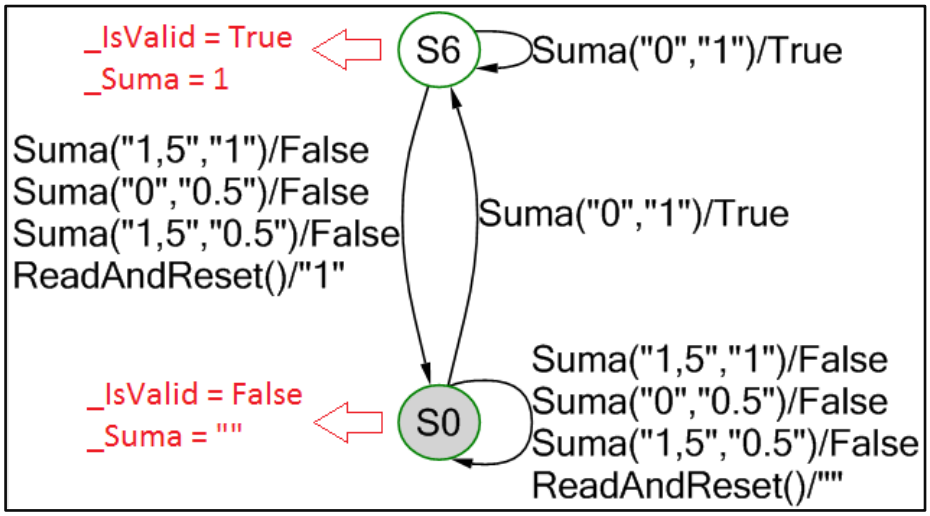
Esta máquina de estados es construida teniendo en cuenta el espacio del dominio limitado, es decir para el conjunto de entradas $x \in \{a, 15\}$ - $y \in \{25,-1\}$, y las acciones establecidas en el modelo. Se puede observar que todas las transiciones resultarán en un estado no valido si alguna de las entradas ingresadas es un texto que no se puede mapear a un número entero mayor o igual a cero.

Figura B-3: Máquina de Estados que modela el comportamiento del Caso de ejemplo para el siguiente dominio de entradas $x \in \{a, 15\}$ - $y \in \{25,-1\}$.



En la Figura B-4 se presenta la máquina de estados para un domino diferente, en este caso *x ∈ {1,5, 0} - y ∈ {0.5, 1}*, para observar el comportamiento con entradas de números decimales en diferentes formatos y el valor cero. Podemos notar que si las entradas no son números enteros mayores o iguales a cero, la operación no se realiza.

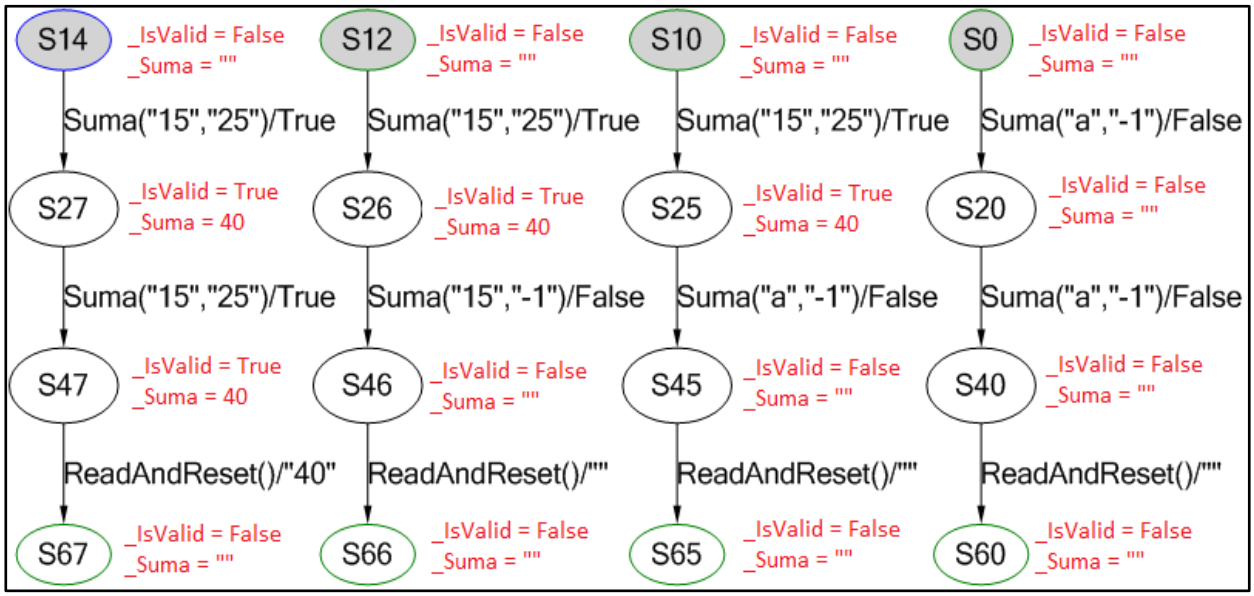
Figura B-4: Máquina de Estados resultante aplicando el siguiente rango de entradas: *x ∈ {1,5, 0} - y ∈ {0.5, 1}.*



B.5 Casos de Prueba

La máquina que define la construcción de los casos de prueba de alto nivel es definida también en el archivo de configuración, la cual es mostrada como una serie de transiciones y estados dependiendo de las acciones que se aplican sobre el modelo (Figura B-5). Esta máquina de pruebas es construida siguiendo la secuencia de dos acciones de suma y una que regresa el valor de la operación a su estado inicial. A continuación se visualiza un segmento de los casos de prueba generados a partir de la máquina de estados de la Figura B-3.

Figura B-5: Máquina de estados con los casos de prueba de alto nivel generados. Se visualizan las acciones aplicadas, los valores de entrada y los resultados esperados.



La herramienta genera todas las transiciones para todas las combinaciones posibles dependiendo del dominio limitado configurado (en nuestro caso $x \in \{a, 15\}$ - $y \in \{25,-1\}$), y siempre al final aplica un re-set del estado retornando el actual. A partir de este se generan los casos de prueba ejecutables con sus entradas, resultados esperados y las transiciones de estados que debería tener la implementación.

B.6 Resultados

El código de pruebas es generado en el proyecto TestViewGUI.TestSuite, un caso de prueba generado es mostrado en el siguiente código, este es solo un caso de los 10 que se generan a partir de la máquina de estados de la Figura B-5.

Test Starting in S10

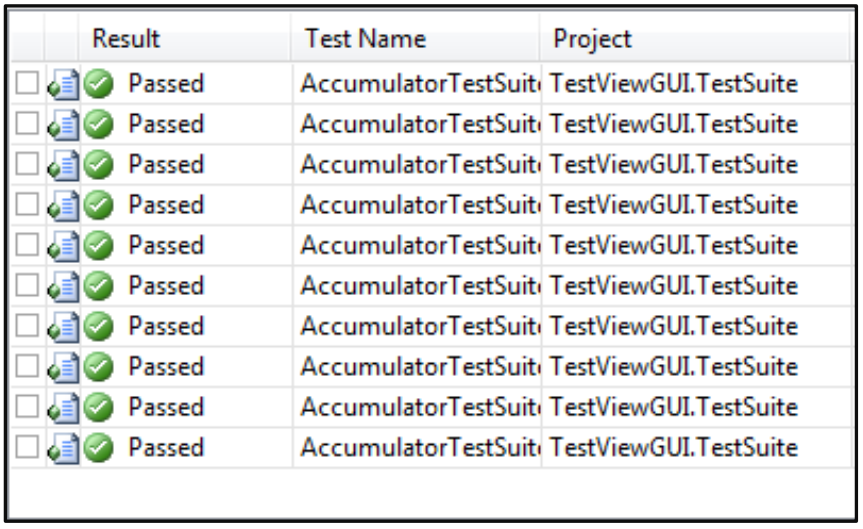
```
public void AdderTestSuiteS10()
{
        this.Manager.BeginTest("AdderTestSuiteS10");
        this.Manager.Comment("reaching state \'S10\'");
        bool temp3;
        this.Manager.Comment("executing step \'call Suma(\"15\",\"25\")\'");
        temp3 = TestViewGUI.Sample.Adder.Suma("15", "25");
        this.Manager.Comment("reaching state \'S11\'");
```

```
        this.Manager.Comment("checking step \'return Suma/True\'");
        TestManagerHelpers.AssertAreEqual<bool>(this.Manager, true, temp3, "return of Suma, state S11");
        this.Manager.Comment("reaching state \'S25\'");
        bool temp4;
        this.Manager.Comment("executing step \'call Suma(\"a\",\"-1\")\'");
        temp4 = TestViewGUI.Sample.Adder.Suma("a", "-1");
        this.Manager.Comment("reaching state \'S35\'");
        this.Manager.Comment("checking step \'return Suma/False\'");
        TestManagerHelpers.AssertAreEqual<bool>(this.Manager, false, temp4, "return of Suma, state S35");
        this.Manager.Comment("reaching state \'S45\'");
        string temp5;
        this.Manager.Comment("executing step \'call ReadAndReset()\'");
        temp5 = Adder.ReadAndReset();
        this.Manager.Comment("reaching state \'S55\'");
        this.Manager.Comment("checking step \'return ReadAndReset/\"\"\'");
        TestManagerHelpers.AssertAreEqual<string>(this.Manager, "", temp5, "return of ReadAndReset, state
S55");
        this.Manager.Comment("reaching state \'S65\'");
        this.Manager.EndTest();
}
```

El anterior código es generado automáticamente para el caso de Prueba S10. Este realiza dos llamados secuenciales a la operación Suma, primero suma *15* y *25*, en lo cual el estado es válido, luego suma *a* y *-1*, en lo cual el resultado es invalido, y por último efectúa un re-set del estado.

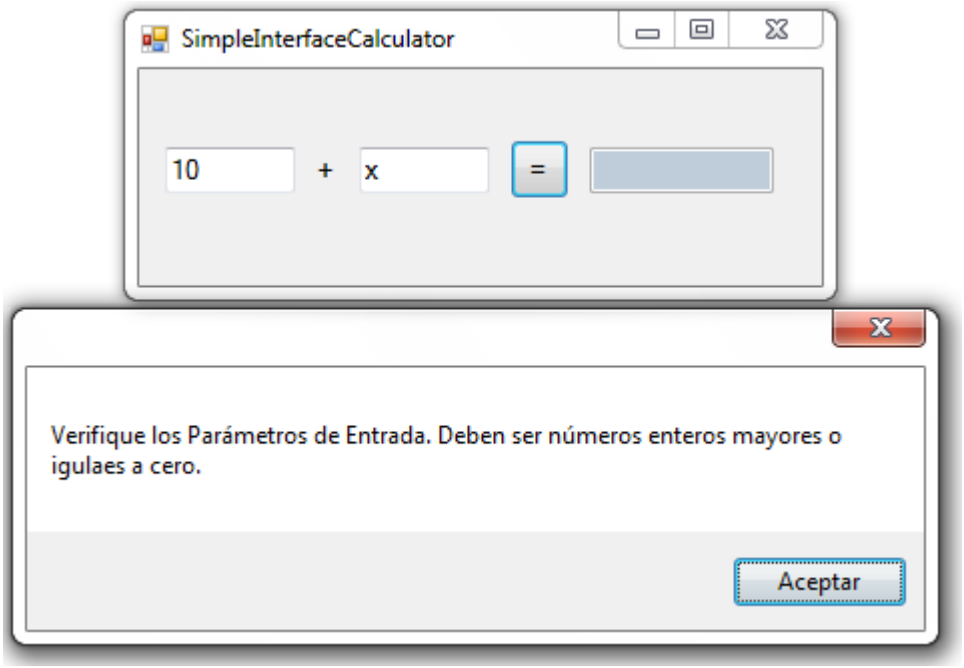
Como podemos observar este código del caso de prueba válida que las salidas sean las esperadas de acuerdo a las entradas y prueba directamente la implementación (en este caso *Adder*). Si la implementación se encuentra acoplada con el modelo, entonces todas las pruebas pasarán. A continuación se presenta la ejecución de pruebas luego de realizar la implementación:

Figura B-6: Ejecución del código de pruebas luego de acoplar la implementación con el modelo. Se generaron 10 casos de pruebas ejecutables.



Esta implementación se conecta con la interfaz gráfica de usuario definida, con el objetivo de verificar que las entradas de usuario sean válidas y la operación de suma sea el esperado.

Figura B-7: Prueba de la interfaz gráfica del sumador luego de conectar la implementación del modelo con la vista.



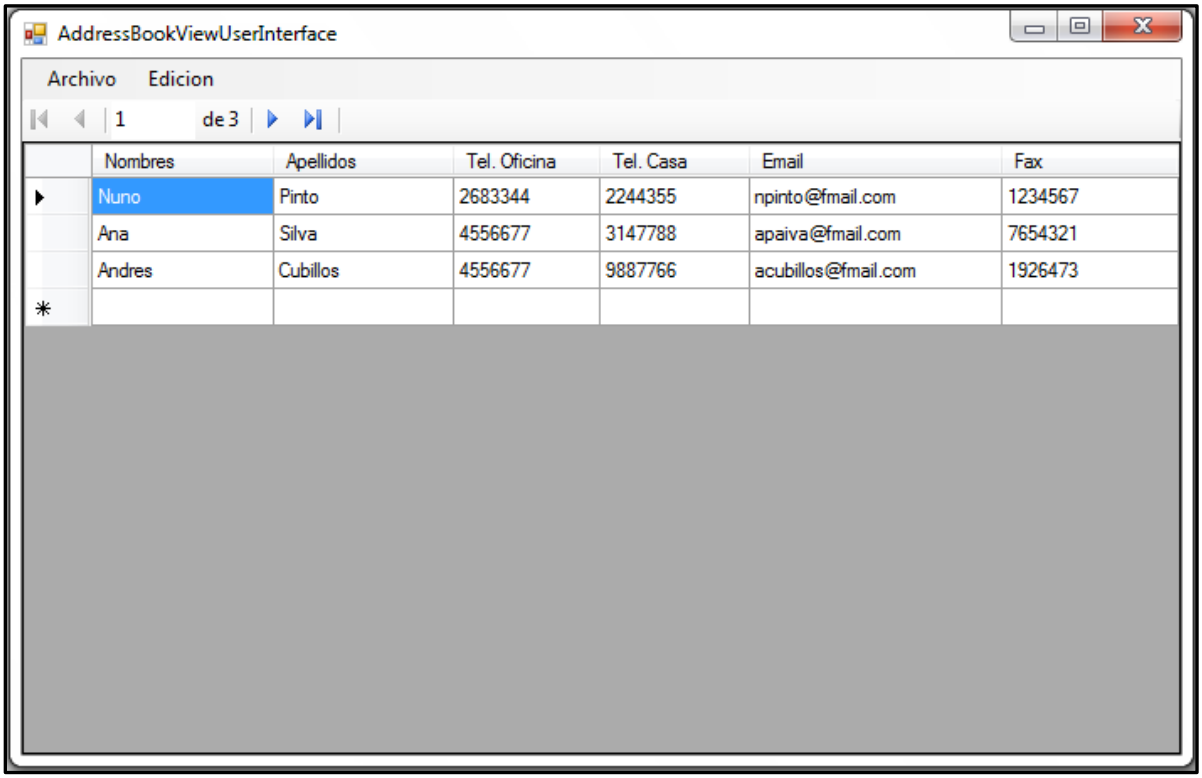
C. Anexo: Implementación del Address Book

El objetivo principal es explorar e implementar la propuesta dada por PAIVA [24] para modelar y automatizar la generación de casos de prueba sobre interfaces graficas de usuario aplicando el proceso de pruebas basadas en modelos empleando la herramienta Spec Explorer de Microsoft. El método presentado en [24] está implementado para interfaces graficas de usuario de estructura jerárquica de aplicaciones desktop y busca probar que las respuestas de la interfaz, cuando el usuario interactúa con ésta, sean las esperadas por los casos de prueba. Como nuestra orientación también es aplicar MBT para GUIs con la herramienta mencionada, se implementó uno de los ejemplos descritos en la tesis de PAIVA como base preliminar para el desarrollo de este trabajo.

C.1 Descripción

El caso de ejemplo implementado es sobre una interfaz de usuario denominada *Address Book* (Libreta de Direcciones, Ver Figura C-8). Esta interfaz básicamente es una lista de contactos de personas e incluye información como nombres, apellidos, teléfonos de contacto, correo electrónico y fax. Como en la mayoría de interfaces comunes tiene las opciones de archivo y edición.

Figura C-8: Interfaz de Usuario Address Book implementada basándose en [24] y [22].

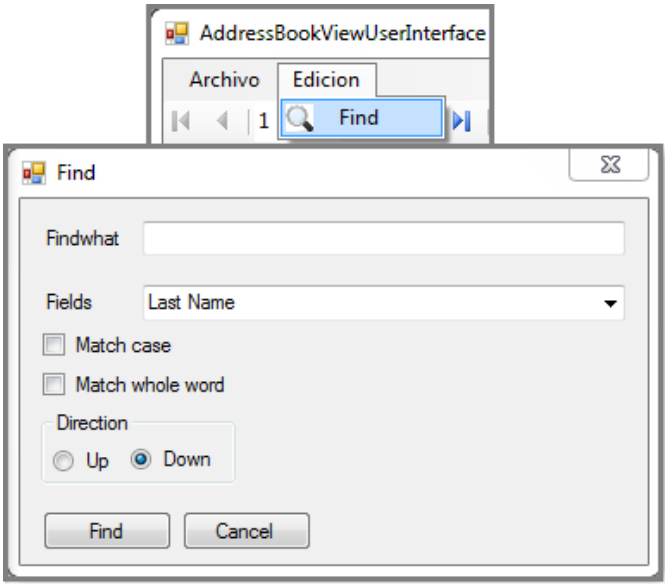


El objetivo de este caso de ejemplo fue probar dos funcionalidades principales:

- ✓ La búsqueda de un nombre en la libreta de direcciones (Find Dialog).
- ✓ La opción de salvar o guardar la libreta actual (Save Dialog).

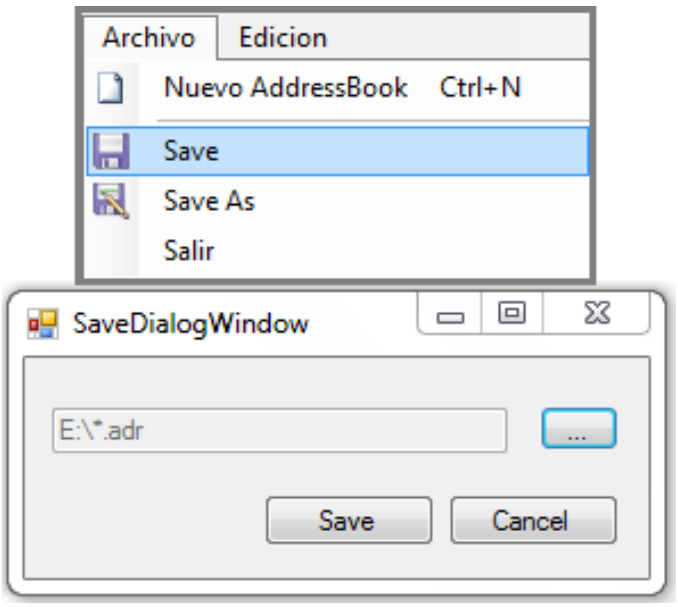
El Find Dialog es mostrado en la Figura C-9 y debe ser desplegado luego de escoger la opción Find del menú Edición. En este diálogo se debe ingresar la palabra que se desea buscar y escoger las opciones de búsqueda entre las que se encuentran el campo, correspondencia con mayúsculas o minúsculas y la dirección búsqueda. De acuerdo al enfoque propuesto en [24] lo que se probará es que se visualice un mensaje al usuario indicando si no se encontró la palabra o se ubique en el ítem correspondiente de la lista en la primera correspondencia.

Figura C-9: Diálogo de Búsqueda de la interfaz (Find Dialog).



El Save Dialog es mostrado en la Figura C-10 y debe ser desplegado luego de escoger la opción Save del menú Archivo. En este diálogo se debe ingresar el path y el nombre del archivo mediante la opción de browse. De acuerdo al enfoque propuesto en [24] lo que se probará es: si el archivo ya existe se visualice un mensaje al usuario indicando si desea sobrescribirlo, sino existe, que se guarde directamente.

Figura C-10: Diálogo de Salvar de la interfaz (Save Dialog).



C.2 Modelado

El enfoque general y el cual será el objetivo de pruebas (según lo expuesto en [24]) es que las ventanas invocadas, las emergentes de alerta y las de confirmación sean las adecuadas cuando el usuario realiza alguna acción. Por este motivo se modelará un estado que indique el nombre de la ventana desplegada según las acciones ejecutadas por el usuario.

El número de estados interno de una interfaz de usuario puede llegar a ser muy grande, de acuerdo con el eventos disparados por el usuario, entonces no resulta una buena estrategia tratar de realizar un solo modelo que represente toda la interfaz, ya que sería muy complejo de manejar, interpretar y mantener. Es por ello que la estrategia recomendada en [24] es realizar un diseño compuesto por diferentes vistas o perspectivas del modelo general que están relacionados a ciertos escenarios. Los escenarios o vistas son obtenidos a partir de las funcionalidades, y por ello para este caso de ejemplo se modelarán dos vistas:

- ✓ FindViewModel: Esta vista contempla el escenario de búsqueda dentro de la interfaz, es decir la secuencia de pasos que un usuario debe realizar para efectuar una búsqueda de una palabra dentro de la lista de contactos empleando el Find Dialog.
- ✓ SaveViewModel: Esta vista contempla el escenario de salvar la lista de contactos actual, es decir la secuencia de pasos que un usuario debe realizar para guardar la lista en un archivo de texto empleando el Save Dialog.

C.1.1 Find View Model

Este modelo pretende modelar el comportamiento de la interfaz desde el punto de vista de realizar una operación de búsqueda sobre el Address Book. Se emulan las acciones que realiza un usuario para realizar dicha operación y se probará que la respuesta dada por la interfaz, bajo la interacción de usuario, sea la esperada.

Las acciones que se modelan básicamente son:

- ✓ *LaunchAddressBook*: Emula la acción de abrir la interfaz del Address Book.

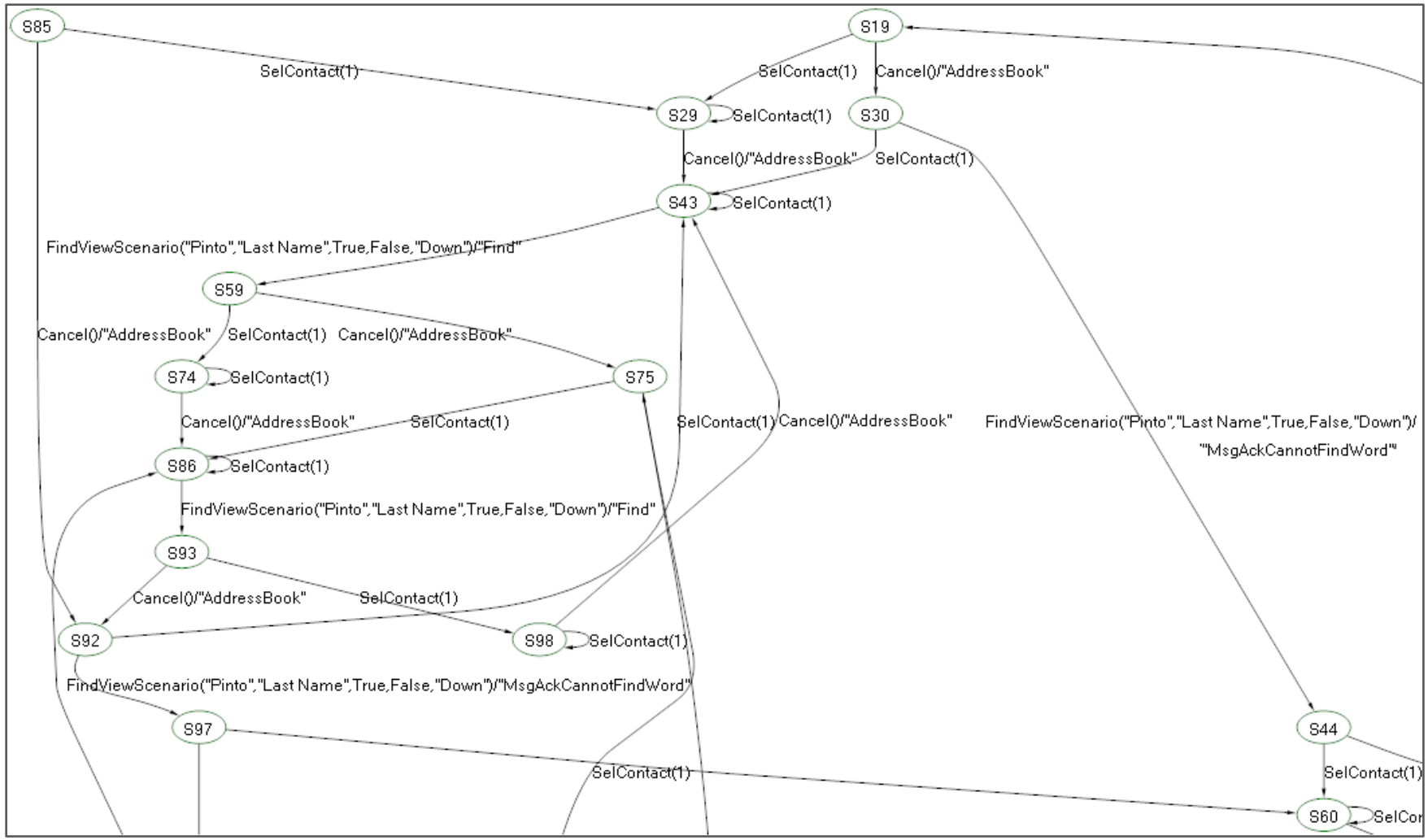
- ✓ *SelContact*: Emula la acción de Seleccionar un ítem de la lista de contactos principal. Tiene como único parámetro de entrada el índice del contacto a seleccionar.
- ✓ *FindViewScenario*: Emula el escenario de realizar una búsqueda por medio del dialogo de búsqueda de la interfaz. Tiene como parámetros de entrada el texto a buscar, el campo o columna en donde realizar la búsqueda, correspondencia con mayúsculas o minúsculas y la dirección búsqueda.
- ✓ *Cancel*: Emula la acción de cancelar el dialogo de búsqueda.

Como el objetivo de esta vista es verificar el estado de las ventanas luego de realizar una operación de búsqueda, el resultado del modelo nos indicará el nombre de la ventana que está activa bajo una acción de usuario.

La exploración del modelo da como resultado una máquina de estados finita que representa los diferentes estados a los que puede llegar la interfaz teniendo en cuenta los eventos externos disparados por el usuario.

En la Figura C-11 se presenta una vista parcial del modelo para el escenario de búsqueda, que es una combinación de las operaciones que participan en el modelo, se puede visualizar el resultado del estado principal de la interfaz, que indica cual ventana esta activa, luego de realizar la acción *FindViewScenario.*

Figura C-11: Exploración de la FSM del Modelo del escenario de Búsqueda.



C.1.2 Save View Model

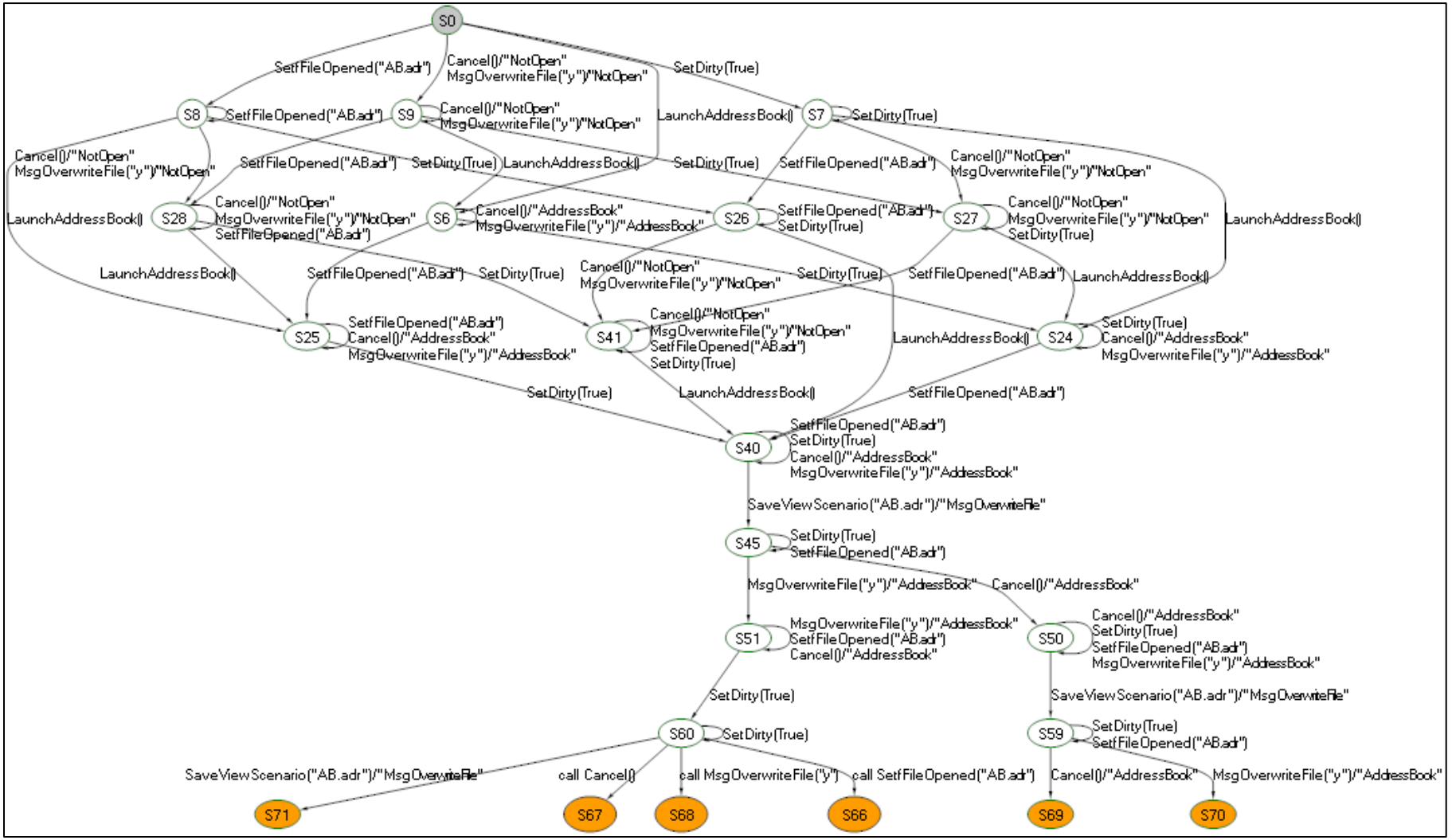
Este modelo pretende modelar el comportamiento de la interfaz desde el punto de vista de realizar una operación de salvar el Address Book actual. Se emulan las acciones que realiza un usuario para realizar dicha operación y se probará que la respuesta dada por la interfaz, bajo la interacción de usuario, sea la esperada.

Las acciones que se modelan básicamente son:

- ✓ *LaunchAddressBook*: Emula la acción de abrir la interfaz del Address Book.
- ✓ *SetDirty*: Emula la acción de establecer la lista de contactos en un estado modificado. Con esto se simula un evento de usuario que modifica algún ítem de la lista indicando que puede ser salvado. Tiene como único parámetro un booleano indicando si la interfaz se ha modificado o no.
- ✓ *SetFileOpened*: Indica el archivo que fue cargado inicialmente y a partir del cual se despliega la información de la lista de contactos. Tiene como único parámetro un string que representa el path completo del archivo cargado.
- ✓ *MsgOverwriteFile:* Si el nombre del archivo con el cual se desea guardar la lista de contactos actual ya existe, este método emula la acción de confirmación del usuario indicando si acepta o no sobrescribir el archivo. Tiene como único parámetro de entrada un string que representa el tipo de confirmación a realizar: “y” para sobrescribir y “n” para cancelar.
- ✓ *SaveViewScenario*: Emula el escenario de realizar un proceso de guardado del libro actual por medio del dialogo de salvar de la interfaz. Tiene como parámetros de entrada el nombre del archivo a guardar.
- ✓ *Cancel*: Emula la acción de cancelar el dialogo de guardar.

En la Figura C-12 se presenta una vista parcial de la máquina de estados finita (representación del modelo) para el escenario de guardar, que es una combinación de las operaciones que participan en el modelo, se puede visualizar el resultado del estado principal de la interfaz que indica cual ventana esta activa, luego de realizar las acciones de *MsgOverwriteFile, SaveViewScenario y Cancel.*

Figura C-12: Exploración de la FSM del Modelo del escenario de Guardado.



C.3 Generación de Pruebas

Para definir los casos de prueba es necesario describir los escenarios de las vistas de los modelos mencionados.

El escenario de búsqueda que se modelará es: Primero abrir AddressBook, luego seleccionar un contacto de la lista, abrir el dialogo de búsqueda, ejecutar una búsqueda con ciertos parámetros de entrada y luego cerrar el diálogo. Se observará la respuesta de la interfaz validando la ventana que está activa.

El escenario de guardar que se modelará es: Primero abrir AddressBook con un archivo inicial, modificar los datos de la lista del libro, abrir el diálogo de guardar, ejecutar una operación de guardar con un path de archivo determinado, si el archivo existe indicar si se sobrescribe o no y por último cancelar el dialogo de guardar. Se observará la respuesta de la interfaz validando la ventana que está activa.

Ahora es necesario definir los valores de prueba de los parámetros de entrada para cada uno de escenarios definidos. En las Tabla C-1 y la Tabla C-2 se presenta la descripción de los valores de prueba que se emplearán en las diferentes acciones que componen los escenarios de prueba modelados: El de Búsqueda y el de Guardar.

Tabla C-1: Datos de Entrada de prueba del escenario de Búsqueda.

Acción		Parámetros de Entrada		Dominio	
LaunchAddressBook	Acción de abrir la interfaz del Address Book.	--	--	--	--
SelContact	Acción de Seleccionar un ítem de la lista de contactos principal.	line	Entero que representa el índice del contacto a seleccionar.	0,1	Se escogerán estos índices de la lista. El valor Cero es un valor inválido.
FindViewScenario	Acción de realizar una búsqueda por medio del dialogo de búsqueda de la interfaz.	fw	Texto a buscar dentro del la lista de contactos.	"Pinto", "Nuno"	Los textos que se buscaran.
		field	Texto que representa el campo o columna por el cual se realizará la búsqueda.	"Last Name"	Solo se buscará por el campo de Last Name.
		mc	Coincidir mayúsculas y minúsculas.	true, false	Se especificará si se desea coincidencia o no.
		mww	Coincidir todo el texto	true, false	Se especificará si se desea coincidencia o no.
		dir	String que representa la dirección de la búsqueda.	"Up", "Down"	Se especificará las dos opciones de dirección de búsqueda.
Cancel	Acción de cancelar el diálogo de búsqueda.	--	--	--	--

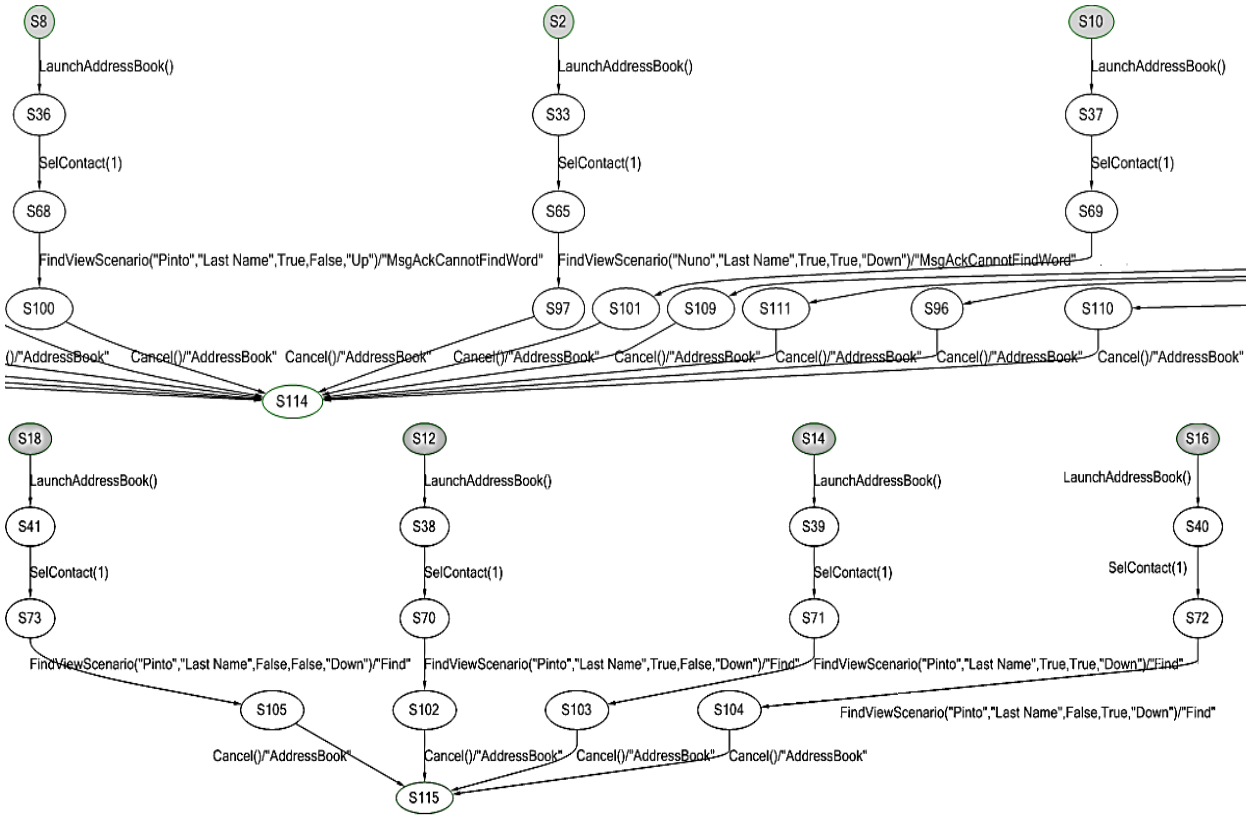
Tabla C-2: Datos de Entrada de prueba del escenario de Guardado.

Acción		Parámetros de Entrada		Dominio	
LaunchAddressBook	Acción de abrir la interfaz del Address Book.	--	--	--	--
SetDirty	Acción de establecer la lista de contactos en un estado modificado.	d	Booleano que indica si está o no modificada la interfaz.	true	Siempre se establecerá la lista de contactos como modificada.
SetFileOpened	Indica el archivo que fue cargado inicialmente.	fileOpened	Texto con el nombre del archivo cargado.	"AB.adr"	Solo habrá un archivo inicialmente cargado.
SaveViewScenario	Acción realizar un proceso de guardado del libro actual por medio del dialogo de salvar de la interfaz	fileName	Texto con el nombre del archivo a guardar.	"AB.adr", "ABwe.adr"	Se establecerán solo dos nombres de archivos.
MsgOverwriteFile	Acción de confirmación del usuario indicando si acepta o no sobrescribir el archivo.	overWrite	Opcion para sobrescribir o no.	"y", "n"	Se escogerán las dos opciones.
Cancel	Acción de cancelar el diálogo de guardar.	--	--	--	--

Definiendo los datos de entrada de pruebas, con la herramienta se generan los casos de prueba abstractos, para el escenario de búsqueda se crearon automáticamente 16 casos de pruebas y para el escenario de guardar 4, que se obtienen de la combinación de datos de entrada para los diferentes contextos.

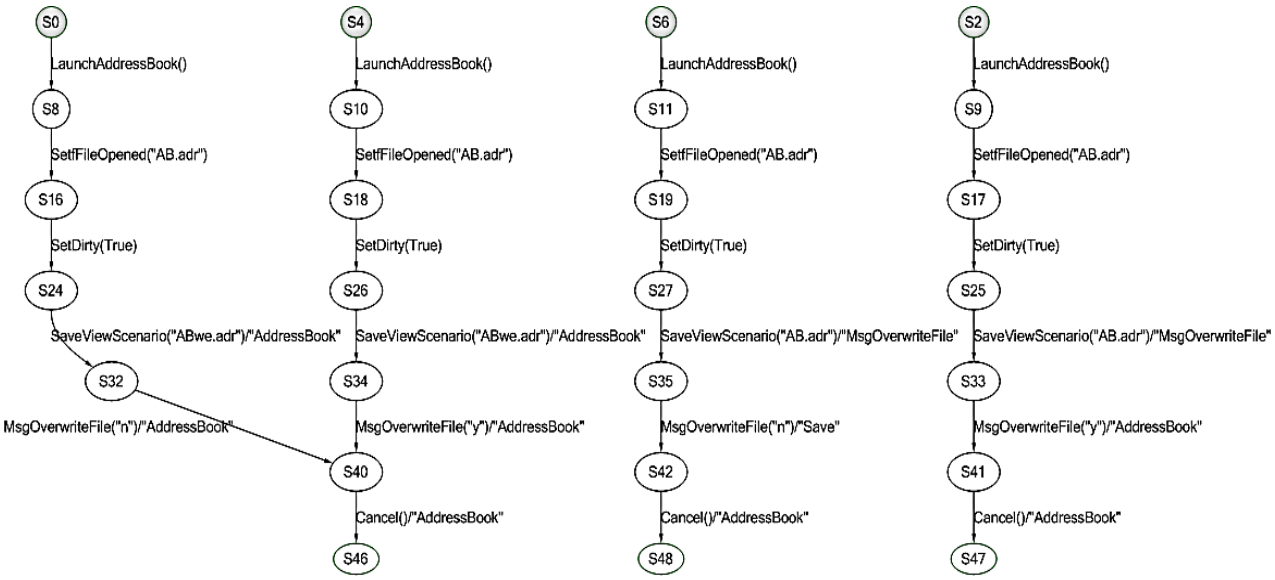
En la Figura C-13 se presentan siete casos de prueba generados automáticamente para el escenario de búsqueda, se puede visualizar los parámetros de entrada de las acciones invocadas y los resultados esperados, el resultado de las acciones representa la ventana que debe estar activa luego de generar la acción. Como se puede observar siempre se selecciona el primer item de la lista (SelContact(1)), ya que se estableció en las precondiciones, que el valor seleccionado debe ser mayor a cero.

Figura C-13: Muestra de casos de prueba de alto nivel generados para el escenario de búsqueda.



En la Figura C-13 se presentan cuatro casos de prueba generados automáticamente para el escenario de guardar, al igual que el caso anterior el resultado de las acciones invocadas representa la ventana que debe estar activa luego de ejecutar la acción. Para efectos de pruebas siempre se coloca la interfaz en un estado modificado (SetDirty(true)).

Figura C-14: Casos de prueba de alto nivel generados para el escenario de guardar.



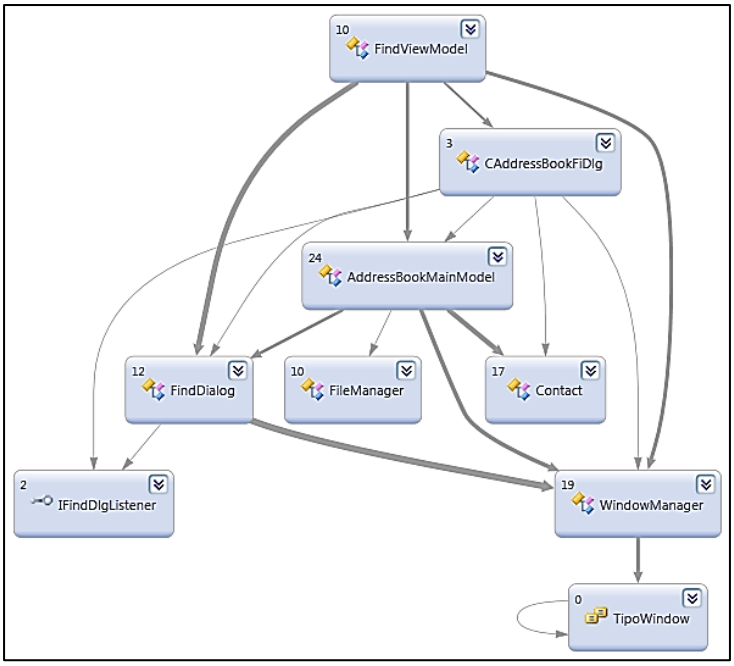
C.4 Implementación

Para diseñar los modelos de pruebas y la implementación con el enfoque propuesto en [24], PAIVA plantea emplear unas clases base tanto en el diseño y la implementación, estas clases las he denominado *Helpers* puesto que contienen funcionalidades que apoyan a las demás clases para lograr los objetivos de prueba y son comodines que pueden ser utilizadas tanto en la fase de diseño del modelo de pruebas como en la implementación.

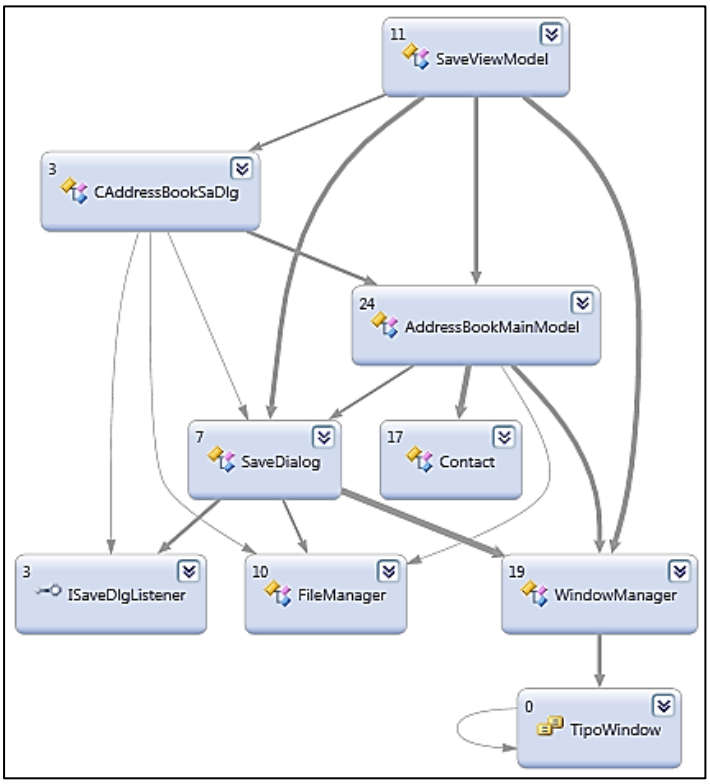
En la Figura C-15 se presenta el diseño general del modelo de pruebas para los dos escenarios del caso de ejemplo. Estos modelos construidos con las facilidades que soporta Spec Explorer.

Figura C-15: Diseño del Modelo de Pruebas para los diferentes escenarios.

a) Modelo de Pruebas para el escenario de Búsqueda.



b) Modelo de Pruebas para el escenario de Búsqueda.



El diseño de los modelos se basa en los helpers: WindowManager, FileManager, SaveDialog y FindDialog, y la implementación de las interfaces IFindDlgListener y ISaveDlgListener. La descripción de los principales elementos es presentado en la siguiente Tabla:

Tabla C-3: Componentes principales del diseño del Modelo general de pruebas.

ELEMENTOS DEL MODELO	
WindowManager	Gestiona en alto nivel y de manera jerárquica las ventanas que componen la interfaz. Controla las ventanas y diálogos que están activas para que pueda interactuar el usuario.
FileManager	Gestiona en alto nivel la interacción con disco. Tiene la referencia de los archivos, con la información de los contactos, almacenados y cargados en la interfaz.
SaveDialog	Gestiona en alto nivel las funcionalidades que componen el escenario de búsqueda.
FindDialog	Gestiona en alto nivel las funcionalidades que componen el escenario de guardar.
IFindDlgListener	Interfaz que expone las funcionalidades explicitas para realizar la búsqueda en la lista de contactos.
ISaveDlgListener	Interfaz que expone las funcionalidades explicitas para poder guardar la lista de contactos en un archivo.

La implementación de las interfaces en la fase de pruebas es de alto nivel, es decir se emula el comportamiento esperado (*CAddressBookSaDlg* y *CAddressBookFiDlg*) y en fase de desarrollo se implementan las acciones concretas. El modelo principal es el AddressBookMainModel y es el que centraliza todas las funcionalidades que debería tener la interfaz, para simplificar este modelo se crean dos perspectivas o vistas según los escenarios de estudio: *FindViewModel* y *SaveViewModel*.

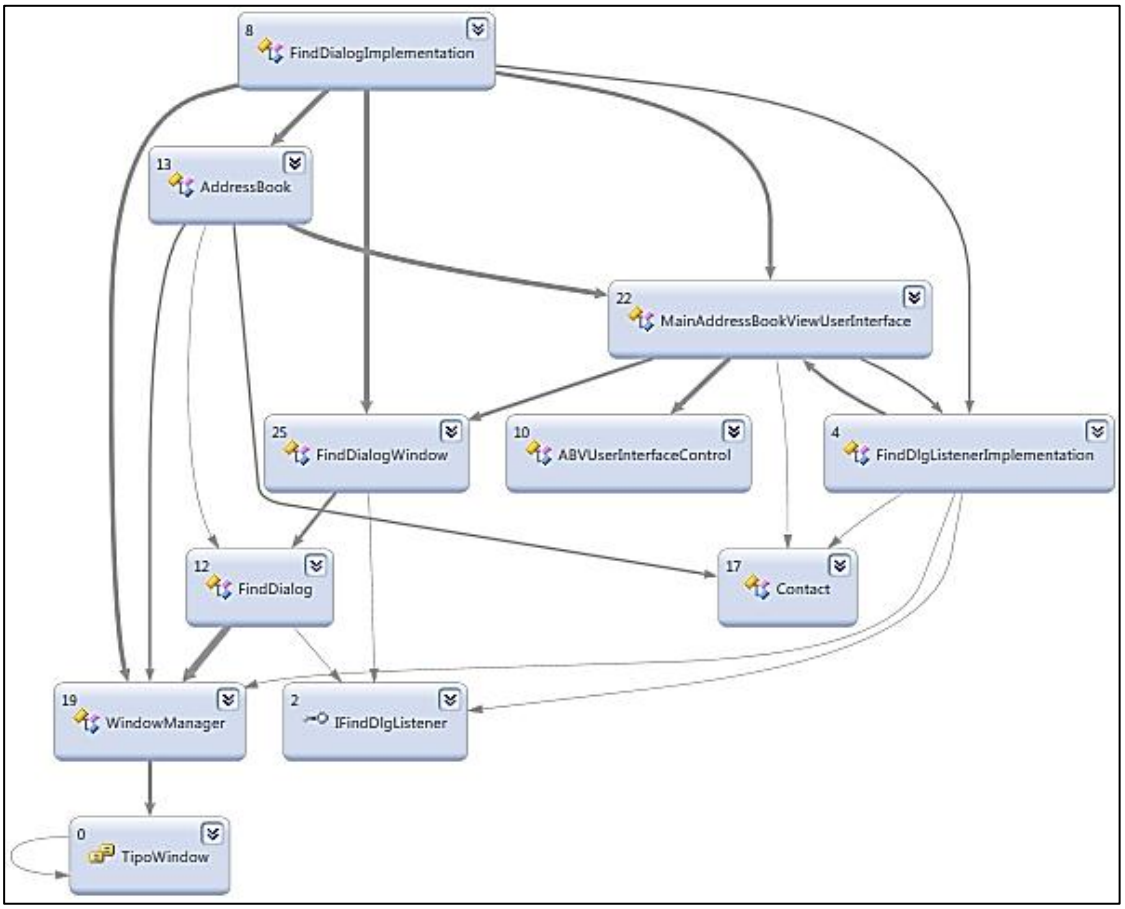
En la Figura C-16 se presenta el diseño general de la implementación concreta para los dos escenarios, como se puede apreciar poseen una estructura similar a la de los modelos, teniendo en cuenta que son una extensión de los mismos. Al igual que en los modelos, se emplean los helpers y la abstracción del modelo principal es el *AddressBook*, el cual es el intermediador y manager para acceder al resto de las

funcionalidades y componentes. También se implementan los escenarios que accederán al intermediador principal y las interfaces que realizarán las operaciones concretas de éstos: *FindDialogImplementation*, *SaveDialogImplementation*, *FindDlgListenerImplementation* y *SaveDlgListenerImplementation*.

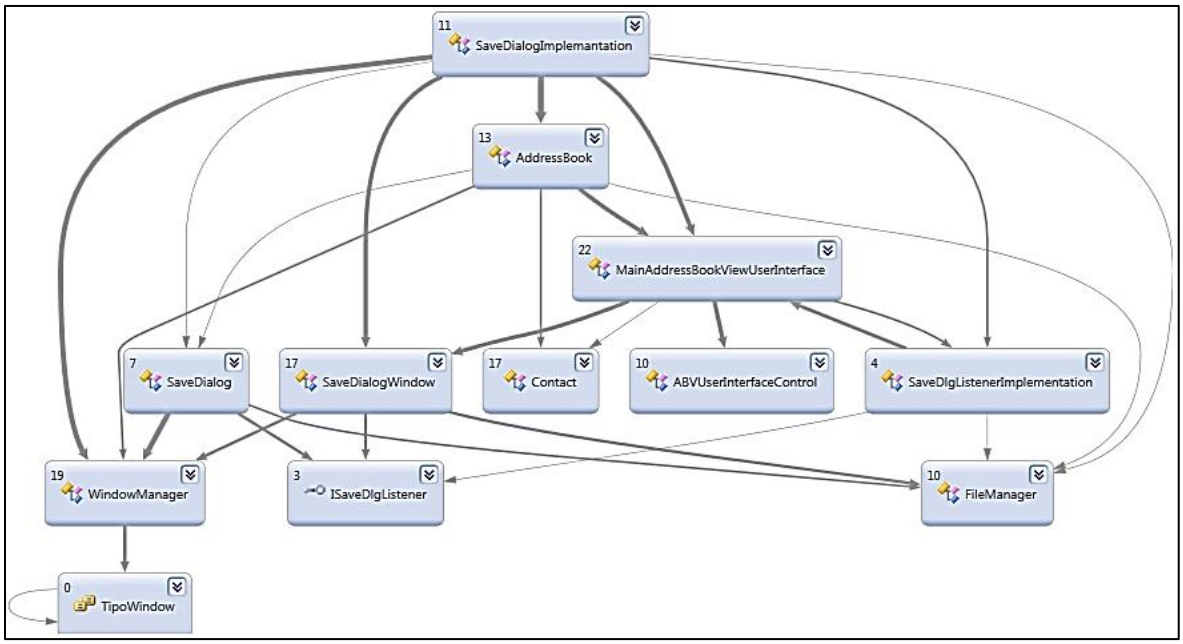
La interfaz gráfica de usuario principal es el *MainAddressBookViewInterface* (Figura C-8) y es a través de ésta que se pueden acceder a los diálogos de Búsqueda y Guardar (*FindDialogWindow* -Figura C-9 y SaveDialogwindow - Figura C-10). En la implementación de pruebas sólo se puede acceder a esta interfaz por medio del manager principal (El *AddressBook*).

Figura C-16: Diseño de la implementación concreta del AddressBook para los diferentes escenarios.

a) Vista de la implementación para el escenario de Búsqueda.



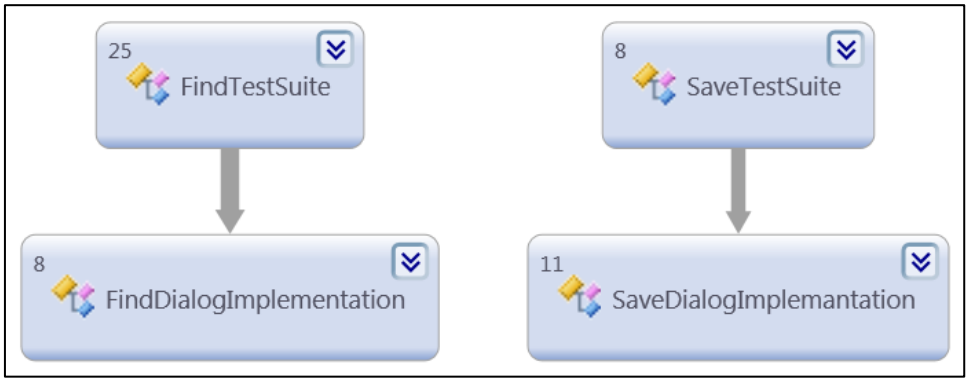
b) Vista de la implementación para el escenario de Guardar.



C.5 Resultados

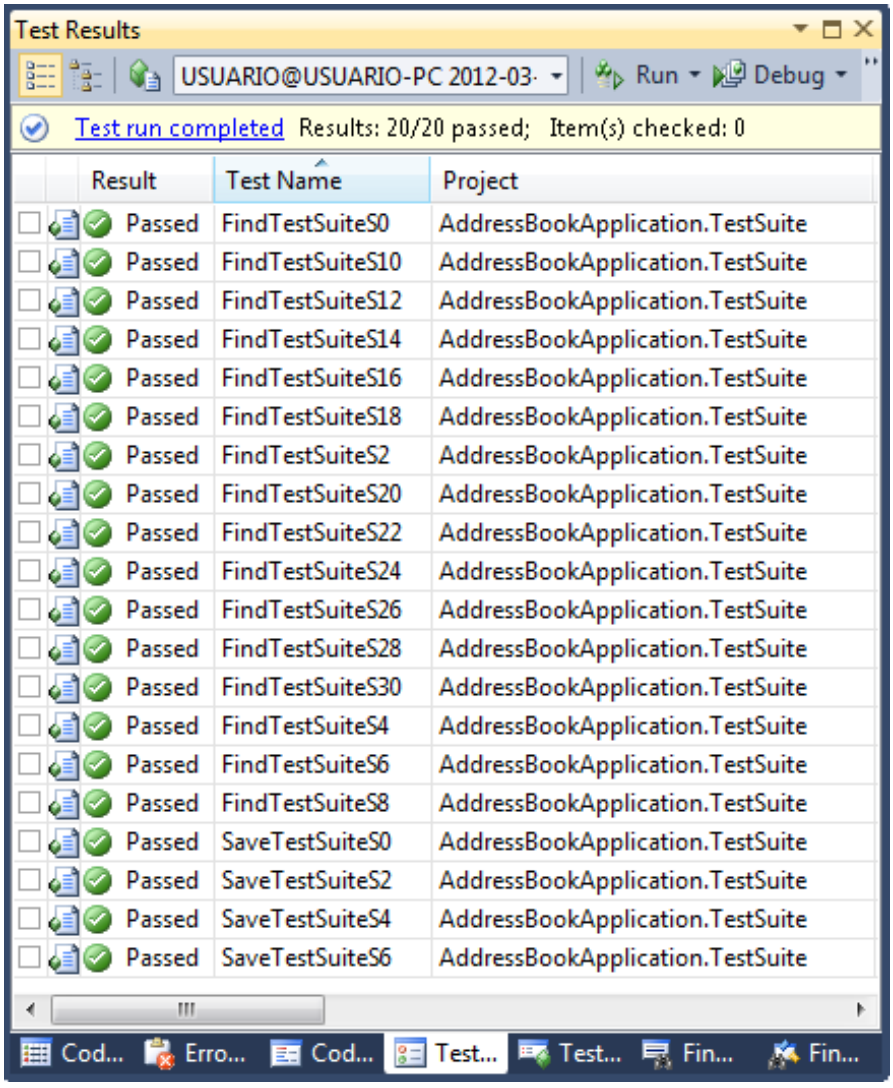
Con la herramienta se generan los casos de pruebas ejecutables que apuntan directamente a la implementación, en este caso el código de pruebas generado automáticamente se establece en los componentes FindTestSuite y SaveTestSuite que prueban explícitamente los escenarios del caso de ejemplo (Figura C-17) .

Figura C-17: Relación entre las pruebas ejecutables y la implementación.



Con la descripción de los valores de entrada de los casos de prueba se generan en total 20 Tests, que prueban el comportamiento de la interfaz desde el punto de vista de saber cuál ventana o dialogo debe estar activo luego de realizar una acción de usuario, si la implementación esta acoplada con el modelo todas las pruebas pasaran (Figura C-18).

Figura C-18: Ejecución del código de pruebas luego de acoplar la implementación con el modelo. Se generaron 20 casos de pruebas ejecutables para dos escenarios propuestos.



D. Anexo: Employers Manager

El objetivo principal es desarrollar un ejemplo en el que se implemente el proceso de pruebas basadas en modelos sobre una interfaz gráfica de usuario compuesta aplicando el enfoque propuesto en este trabajo y empleando la herramienta *Spec Explorer 2012* para especificar el comportamiento del modelo. El enfoque se basa en los siguientes aspectos para modelar y probar la interfaz:

- ✓ Emplear el patrón MVP (*Model View Presenter)* como arquitectura base para diseñar el comportamiento del modelo y construir la implementación.
- ✓ Dividir el comportamiento de la interfaz principal en sub-modelos que representen los escenarios que abstraen los casos de prueba (recomendada en [24]).
- ✓ Probar que los datos que ingresa el usuario en los diferentes campos sean válidos, tanto en longitud como en formato.
- ✓ Probar que el despliegue de ventanas, diálogos y mensajes informativos mostrados por la interfaz, en respuesta a las acciones del usuario, sean los esperados (propuesto en [24]).
- ✓ Probar que un control determinado pueda o no responder a la interacción del usuario de acuerdo a condiciones definidas.
- ✓ Probar la actualización de un control de acuerdo la respuesta de otro.

D.1 Descripción

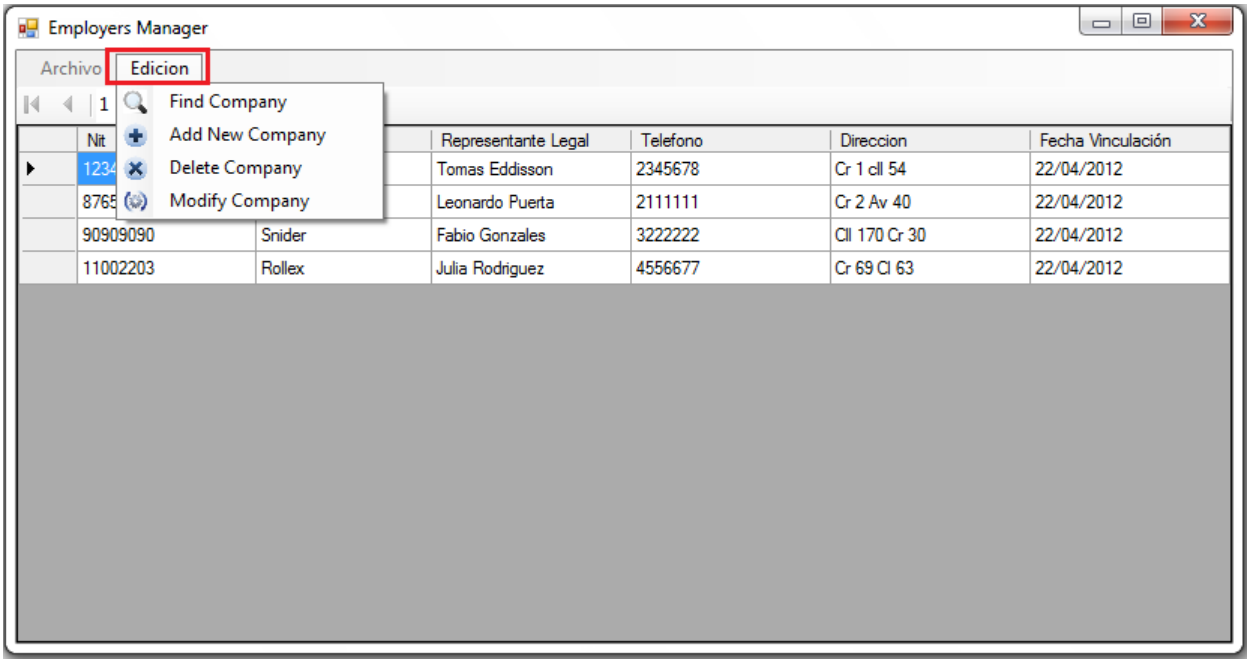
El caso de ejemplo implementado es sobre una interfaz gráfica de usuario parecida a la del anexo anterior pero en este caso se almacena la información de empleadores o empresas afiliadas a una caja de compensación y se ha denominado *Employers Manager* (ver Figura D-19). Esta interfaz básicamente es una lista de empresas, y de cada una se almacena la siguiente información:

- ✓ Nit de la empresa.
- ✓ Nombre de la empresa.
- ✓ Representante legal de la empresa.
- ✓ Teléfono.
- ✓ Dirección.
- ✓ Fecha de Vinculación a una caja de compensación familiar.

La interfaz expone cuatro funcionalidades principales por medio del menú de edición:

- ✓ Adicionar una nueva empresa.
- ✓ Modificar una empresa existente.
- ✓ Eliminar una empresa.
- ✓ Buscar una empresa.

Figura D-19: Interfaz Gráfica de Usuario Employers Manager (Vista Principal).



D.2 Casos de Uso

Para formalizar el caso de ejemplo se presenta la especificación de los requerimientos y casos de uso principales de la interfaz gráfica de usuario con una plantilla sencilla e intuitiva (ver Tabla D-4, Tabla D-5, Tabla D-6 y Tabla D-7). La descripción de requerimientos es la base para diseñar el comportamiento del modelo de la interfaz gráfica de usuario.

Tabla D-4: Especificación del caso de uso: Agregar nueva empresa.

Nombre	**CUS – 1. Agregar nueva Empresa**
Actores	Interfaz gráfica de usuario Employers Manager, Usuario.
Sinopsis	Este caso de uso inicia cuando el usuario selecciona en el menú edición el botón adicionar nueva compañía
Escenario típico	
	1. El usuario inicia la interfaz gráfica Employers Manager. 2. Dentro de la vista compuesta Employers Manager, el usuario selecciona en el menú edición el botón agregar nueva compañía. 3. Se despliega el formulario respectivo al ingreso de una nueva compañía. Dentro del mismo aparecen los campos respectivos de ingreso: a. Nit (numérico, >= 0 y <= 99999999999999, no debe existir en la lista y es obligatorio). b. Nombre empresa (cadena caracteres <= 50, obligatorio). c. Representante legal (cadena de caracteres < =50, obligatorio). d. Dirección (cadena caracteres <= 50, obligatorio), e. Teléfono (cadena caracteres <= 15, obligatorio) , f. Fecha vinculación (date <= fecha actual, obligatorio). 4. La interfaz valida el contenido de los datos. a. Si alguno de los campos no se ingresa correctamente no se debe activar el botón guardar. b. Adicionalmente se debe visualizar una advertencia sobre el campo o los campos que no se ingresaron correctamente. c. La interfaz valida si todos los campos fueron ingresados. d. Si todos los datos fueron ingresados y además se ingresan correctamente se activa el botón guardar. 5. El usuario selecciona el botón guardar una vez este haya sido activado. 6. Se visualiza un mensaje indicando que el proceso se realizó correctamente. 7. El formulario de ingreso queda abierto hasta que el usuario cierre la ventana respectiva. 8. El usuario queda dentro de la vista compuesta Employers Manager.
Escenarios alternativos	

	1. Información de algún dato no válido, teniendo en cuenta el formato respectivo. a. Un mensaje de alerta es activado para cada campo que presente el problema. b. El formulario de edición de datos no permite guardar la información dejando inactivo el botón guardar. c. El usuario cancela el formulario y no guarda la información.

Tabla D-5: Especificación del caso de uso: Modificar datos empresa.

Nombre	**CUS – 2.Modificar datos empresa.**
Actores	Interfaz gráfica de usuario Employers Manager, Usuario.
Sinopsis	Este caso de uso inicia cuando el usuario selecciona en el menú edición el botón modificar compañía.
Escenario típico	
	1. El usuario inicia la interfaz gráfica Employers Manager. 2. El usuario selecciona un registro del listado respectivo de empresas. 3. El usuario selecciona en el menú edición el botón modificar compañía. 4. Se despliega el formulario de edición de datos de la compañía seleccionada. Dentro del mismo aparecen los campos respectivos de ingreso: a. Nit (numérico, >= 0 y <= 99999999999999, este campo debe aparecer inhabilitado para edición y es obligatorio). b. Nombre empresa (cadena caracteres < 50, obligatorio). c. Representante legal (cadena de caracteres < 50, obligatorio). d. Dirección (cadena caracteres < 50, obligatorio), e. Teléfono (cadena caracteres < 15, obligatorio), f. Fecha vinculación (date <= fecha actual, obligatorio). 5. La interfaz valida el contenido de los datos. a. Si alguno de los campos no se ingresa correctamente no se debe activar el botón guardar. b. Adicionalmente se debe visualizar una advertencia sobre el campo o los campos que no se ingresaron correctamente. c. La interfaz valida si todos los campos fueron ingresados. d. Si todos los datos fueron ingresados y además se ingresan correctamente se activa el botón guardar. 6. El usuario selecciona el botón guardar una vez este haya sido activado. 7. Se visualiza un mensaje indicando que el proceso se realizó correctamente. 8. El formulario de edición queda abierto hasta que el usuario cierre la ventana respectiva. 9. El usuario queda dentro de la vista compuesta Employers Manager.
Escenarios alternativos	

	1. Información de algún dato no válido, teniendo en cuenta el formato respectivo. a. Un mensaje de alerta es activado para cada campo que presente el problema. b. El formulario de edición de datos no permite guardar la información dejando inactivo el botón guardar. c. El usuario cancela el formulario y no guarda la información.

Tabla D-6: Especificación del caso de uso: Borrar datos empresa.

Nombre	**CUS – 3.Eliminar empresa.**
Actores	Interfaz gráfica de usuario Employers Manager, Usuario.
Sinopsis	Este caso de uso inicia cuando el usuario selecciona en el menú edición el botón borrar compañía.
Escenario típico	
	1. El usuario inicia la interfaz gráfica Employers Manager. 2. El usuario selecciona un registro del listado de empresas. 3. El usuario selecciona en el menú edición el botón borrar compañía. 4. El registro desaparece de la lista en la interfaz principal.

Tabla D-7: Especificación del caso de uso: Buscar empresa.

Nombre	**CUS – 4.Buscar empresa**
Actores	Interfaz gráfica de usuario Employers Manager, Usuario.
Sinopsis	Este caso de uso inicia cuando el usuario selecciona en el menú edición el botón buscar compañía.
Escenario típico	

	1. El usuario inicia la interfaz gráfica Employers Manager. 2. Dentro de la vista compuesta Employers Manager, el usuario selecciona en el menú edición el botón buscar compañía. 3. Se despliega el cuadro de diálogo respectivo para buscar una compañía. Dentro del mismo aparecen los campos de los parámetros de búsqueda: a. FindWhat: Texto que se desea buscar. Menor a 20 caracteres y es obligatorio. b. Field: Indica si se busca por nit, nombre o representante legal; por defecto se busca por nombre. c. Match Case (coincidir mayúsculas y minúsculas): Indica que en la búsqueda deben coincidir mayúsculas y minúsculas. 4. La interfaz valida el contenido de los datos. a. Si el campo FindWhat no se ingresa correctamente no se activa el botón buscar. b. Se visualiza una advertencia sobre el campo que no se ingresó correctamente. c. La interfaz valida si todos los campos fueron ingresados. d. Si el valor de búsqueda se ingresa correctamente se activa el botón buscar. 5. El usuario selecciona el botón buscar una vez este haya sido activado. 6. Si se encuentran datos, es resaltado el registro en la primera coincidencia de arriba hacia abajo dentro del listado de la interfaz principal. 7. Si no hubo resultados de la búsqueda, aparece un mensaje informativo. 8. El usuario queda dentro de la vista compuesta del Employers Manager.
Escenarios alternativos	
	1. Información de algún dato no válido, teniendo en cuenta el formato respectivo. a. Un mensaje de alerta es activado para el campo de búsqueda. b. El dialogo de búsqueda no permite buscar la información dejando el botón inactivo. c. El usuario cancela el dialogo y no busca información.

D.3 Casos de Prueba

Para complementar los casos de uso es necesario especificar los casos de prueba, estos serán la base para diseñar el comportamiento de los escenarios en la fase de modelado, además servirán para ejecutar las pruebas manuales (validación de usuario final) que es el proceso de referencia para validar el método empleado.

Para cada caso de uso se ha definido un caso de prueba exitoso y uno no exitoso, que la interfaz gráfica de usuario debe implementar para garantizar que se están cumpliendo con los objetivos de los requerimientos iniciales. Los casos de pruebas son mostrados a continuación:

Tabla D-8: Casos de prueba para el escenario de: Agregar una nueva empresa.

a) Caso de prueba exitoso.

Nombre	**CP1-1.Ingresar una nueva compañía con datos correctos.**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02 -> 03 (nit,nombre empresa,representante legal,dirección,telefono,fecha vinculación) -> 04 -> 05 -> 06 -> 07 01 Abrir la interfaz principal, vista compuesta Employers Manager. 02 Desde el menú edición seleccionar el botón de ingreso de una nueva compañía. 03 El sistema visualiza el formulario correspondiente para ingreso de los datos de la compañía. 04 El sistema valida el contenido de los campos, si alguno de estos no se ha ingresado o no esta en el formato adecuado el sistema no activa la opción de guardar. 05 El sistema guarda la información de la compañía generando un mensaje de proceso exitoso. 06 El usuario cierra el formulario de ingreso de datos. 07 El usuario cierra la interfaz principal.
Valores de prueba	nit = “999999999999999” nombre empresa="AYAX LTDA." representante legal="Tomas Edisson" dirección="Cr 5 Av 4" teléfono="2233445" fecha vinculación="2012-01-31"
Resultados observables	01 -> 02 -> 03 [V03] -> 04 -> 05 -> 06 -> 07
	V03, El sistema muestra el formulario para ingresar la información de la compañía.

b) Caso de prueba no exitoso.

Nombre	**CP1-2.Ingresar una nueva compañía con datos incorrectos.**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02 -> 03(nit,nombre empresa,representante legal,dirección,telefono,fecha vinculación) -> 04 -> 05 -> 06 -> 07 01 Abrir la interfaz principal, vista compuesta Employers Mamager. 02 Desde el menú edición el usuario selecciona el botón de ingreso de una nueva compañía. 03 El sistema visualiza el formulario correspondiente para ingreso de los datos de la compañía. 04 El sistema valida el contenido de los campos, si alguno de estos no se ha ingresado o no esta en el formato adecuado el sistema no activa la opción de guardar. 05 El sistema no permite guardar la información y el usuario cierra el formulario de ingreso de datos. 06 El usuario cierra la interfaz principal.

Valores de prueba	nit = “12345678”, nit = “letras” nombre empresa=" " representante legal=" " dirección=" " teléfono="1234567890123451" fecha vinculación="2012-12-31"
Resultados observables	01 -> 02 -> 03 [V03]-> 04 -> 05 -> 06
	V03, El sistema visualiza el formulario para ingresar la información de la compañía.

Tabla D-9: Casos de prueba para el escenario de: Modificar una empresa existente.

a) Caso de prueba exitoso.

Nombre	**CP2-1Modificar una empresa con datos correctos para su ingreso exitoso.**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02 (item) -> 03 (nit, nombre empresa, representante legal, dirección, teléfono, fecha vinculación) -> 04 -> 05 -> 06 -> 07 -> 08 01 Abrir la interfaz principal, vista compuesta Employers Manager. 02 El usuario selecciona un campo de un registro dentro del listado respectivo de empresas (selecciona una compañía o item de la lista). 03 Desde el menú edición seleccionar el botón de modificación de la compañía. 04 El sistema visualiza el formulario correspondiente para actualización de los datos de la compañía, dicho formulario visualiza la información actual de la compañía, el único campo que no se puede modificar es el nit, pues aparece inhabilitado. 05 El sistema valida el contenido de los campos actualizados, si alguno de estos no se ha ingresado o no esta en el formato adecuado el sistema no activa la opción de guardar. 06 El sistema actualiza la información de la compañía generando un mensaje de proceso exitoso. 07 El usuario cierra el dialogo respectivo al formulario de actualización de datos. 08 El usuario cierra la interfaz principal.
Valores de prueba	nombre empresa="FENIX" representante legal="Campo Elias" dirección="Cr 6 Av 4" teléfono="7654321" fecha vinculación="2012-01-31" item=0
Resultados observables	01 -> 02 [V02] -> 03 [V03] -> 04 -> 0 5 -> 06 -> 07 -> 08
	V02, El usuario selecciona la compañía o item dentro de la lista en la vista principal.
	V03, El sistema muestra el formulario para modificar la información de la compañía.

b) Caso de prueba no exitoso.

Nombre	**CP2-2.Modificar una empresa con datos incorrectos.**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02 (item) -> 03 (nit, nombre empresa, representante legal, dirección, teléfono, fecha vinculación) -> 04 -> 05 -> 06 -> 07 01 Abrir la interfaz principal, vista compuesta Employers Manager. 02 El usuario selecciona un campo de un registro dentro del listado respectivo de empresas (selecciona una compañía o item de la lista). 03 Desde el menú edición seleccionar el botón de modificación de la compañía. 04 El sistema visualiza el formulario correspondiente para actualización de los datos de la compañía, dicho formulario visualiza la información actual de la empresa, el único campo que no se puede modificar es el nit, pues aparece inhabilitado. 05 El sistema valida el contenido de los campos actualizados, si alguno de estos no se ha ingresado o no esta en el formato adecuado el sistema no activa la opción de guardar. 06 El usuario no puede modificar la información y cierra el dialogo respectivo al formulario de actualización de datos. 07 El usuario cierra la interfaz principal.
Valores de prueba	nombre empresa="12345678901234567890123456789012345678901234567890123456789012" representante legal="12345678901234567890123456789012345678901234567890123456789012" dirección="12345678901234567890123456789012345678901234567890123456789012" teléfono="1234567890123456" fecha vinculación="2012-12-31" item=1
Resultados observables	01 -> 02 [V02] -> 03 [V03] -> 04 -> 05 -> 06 -> 07
	V02, El usuario selecciona la compañía o item dentro de la lista principal.
	V03, El sistema visualiza el diálogo para modificar la información de la compañía.

Tabla D-10: Casos de prueba para el escenario de: Eliminar una empresa.

a) Caso de prueba exitoso.

Nombre	**CP3-1.Borrar una compañía exitosamente**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02 (item) -> 03 -> 04 -> 05 -> 06 (item) -> 07 01 Abrir la interfaz principal, vista compuesta Employers Manager. 02 Selecciona un registro dentro del listado respectivo de empresas (seleccionar una compañía o item de la lista), por defecto aparece seleccionado el primer item. 03 Desde el menú edición, el usuario selecciona el botón de eliminación de la compañía. 04 El sistema borra la compañía seleccionada. 05 La compañía o item no aparece en la lista de la vista principal. 06 El usuario selecciona un registro dentro del listado respectivo de empresas. 07 Desde el menú edición, el usuario selecciona el botón de eliminación de la compañía. 08 El sistema borra la compañía seleccionada. 09 La compañía o item no aparece en la lista dentro de la vista principal. 10 El usuario cierra la interfaz principal.
Valores de prueba	item="0" item="1"
Resultados observables	01 -> 02 [V02] -> 03 -> 04 -> 05 -> 06[V06] -> 07 -> 08 -> 09 ->10
	V02, El usuario selecciona la compañía o item dentro de la lista principal.
	V06, El usuario selecciona la compañía o item dentro de la lista principal.

b) Caso de prueba no exitoso.

Nombre	**CP3-2.Intentar borrar una compañía cuando no existen datos en la lista.**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02 -> 03 01 Abrir la interfaz principal, vista compuesta Employers Manager. 02 Desde el menú edición el usuario intenta seleccionar el botón de eliminación de la compañía pero está inhabilitado. 03 El usuario cierra la interfaz principal.
Valores de prueba	item= vacío
Resultados observables	01 -> 02 -> 03

Tabla D-11: Casos de prueba para el escenario de: Buscar una empresa

a) Caso de prueba exitoso.

Nombre	**CP4-1.Buscar una empresa con datos correctos.**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02(item) -> 03 -> 04 (FindWhat, Field, MatchCase) ->05 ->06 ->07->08 01 Abrir la interfaz principal, vista compuesta Employers Manager. 02 El usuario selecciona un registro dentro del listado respectivo de empresas. 03 Desde el menú edición el usuario selecciona el botón de búsqueda de la compañía. 04 El sistema visualiza el dialogo correspondiente para búsqueda de la compañía, dicho diálogo visualiza el campo de texto para el ingreso del valor de búsqueda *(FindWhat),* un check box que indica si la búsqueda debe ser sensible a mayúsculas y minúsculas *(matchCase)* y una lista desplegable con los campos de búsqueda *(Field),* que tiene como valores: nit, nombre y representante legal. Por defecto aparece seleccionado el nombre. 05 El sistema valida el contenido del valor ingresado, si no se ha ingresado o no esta en el formato adecuado, el sistema no activa la opción de búsqueda. 06 El sistema busca la compañía y si hay resultado, se resalta el primer item de coincidencia. 07 El usuario cierra el dialogo de búsqueda. 08 El usuario cierra la interfaz principal.
Valores de prueba	FindWhat="90909090" ,"Rollex","Leonardo" Field="Nit","Nombre", "Representante Legal" matchCase="false" item=0
Resultados observables	01 -> 02 [V02]-> 03 -> 04 [V04] -> 05->06->07->08
	V02, El usuario selecciona la compañía o item dentro de la lista principal.
	V04, El sistema muestra el diálogo de búsqueda.

b) Caso de prueba no exitoso.

Nombre	**CP4-2.Buscar una empresa con datos incorrectos.**
Objetivo	Este caso de prueba verifica el comportamiento del caso de uso, en su escenario principal.
Acciones	01 -> 02(item) -> 03 -> 04 (FindWhat, Field, MatchCase) ->05 ->06 ->07 01 Abrir la interfaz principal, vista compuesta Employers Manager. 02 Selecciona un registro dentro del listado respectivo de empresas. 03 Desde el menú edición seleccionar el botón de búsqueda de la compañía. 04 El sistema visualiza el diálogo correspondiente para búsqueda de la compañía. 05 El sistema valida el valor ingresado; si está vacío o no está en el formato adecuado, el sistema no activa la opción de búsqueda. 06 El sistema visualiza un mensaje informando que no hubo resultados de la búsqueda. 07 El usuario cierra el dialogo de búsqueda. 08 El usuario cierra la interfaz principal.
Valores de prueba	FindWhat="XX" ,"123456789012345678 90123456789012" Field="Nit","Nombre", "Representante Legal" matchCase="false" item=1
Resultados observables	01 -> 02 [V02]-> 03 -> 04 [V04] -> 05 -> 06 ->07-> 08
	V02, El usuario selecciona la compañía o item dentro de la lista principal.
	V04, El sistema muestra el diálogo de búsqueda.

D.4 Modelado

De acuerdo al enfoque descrito al inicio del caso de ejemplo se empleará el patrón MVP (*Model View Presenter)* como arquitectura base para modelar la interfaz y se especificará un modelo por cada escenario de los casos de uso, de tal forma que sea más comprensible el comportamiento general de la interfaz.

Los escenarios modelados son los siguientes:

- ✓ *ScenarioAddCompanyModel*: Modela el escenario de agregar una nueva compañía empleando el formulario de edición de datos de empresa.
- ✓ *ScenarioUpdateCompanyModel*: Modela el escenario de modificar la información de una compañía existente empleando el formulario de edición de datos de empresa.
- ✓ *ScenarioDeleteCompanyModel*: Modela el escenario de eliminar una compañía desde la interfaz principal.

- ✓ *ScenarioFindCompanyModel*: Modela el escenario de búsqueda dentro de la interfaz, empleando el dialogo de búsqueda.

En general para cada escenario se observará del estado general de la interfaz, es decir en la máquina de estados se visualizará el consolidado del estado de las principales variables que componen el comportamiento del modelo.

D.4.1 Scenario: *AddCompanyModel*

Modela el comportamiento de la interfaz basándose en el caso de uso CUS-1 (Tabla D-4) y en los casos de prueba CP1-1 y CP1-2 (Tabla D-8). Se emulan las acciones que ejecuta un usuario para realizar la operación de agregar una nueva empresa y se prueba que la respuesta dada por la interfaz bajo la interacción de usuario sea la esperada. Las acciones modeladas son:

- ✓ *LaunchView*: Emula la acción de abrir la interfaz Employers Manager.
- ✓ *AddNewCompany*: Emula la acción de abrir el formulario de ingreso de datos de la nueva empresa.
- ✓ *CompanyViewScenario*: Emula la acción de usuario de ingresar los datos de la nueva empresa en los campos del formulario. Tiene como parámetros de entrada el Nit, el Nombre, el Representante Legal, el Teléfono, la Dirección y la Fecha de Vinculación de la empresa.
- ✓ *Guardar*: Emula la acción de oprimir el botón de guardar del formulario.
- ✓ *MsgAckProceesOk*: Emula la respuesta de la vista indicando que el proceso de guardar la información se realizó correctamente.
- ✓ *Cancel*: Emula la acción de cancelar el formulario de ingreso de datos de la nueva empresa.
- ✓ *Close:* Emula la acción de cerrar la interfaz principal.

Como el objetivo de este modelo es verificar el estado de la interfaz en un punto determinado de ejecución, se observarán las siguientes variables que detallan su comportamiento:

- ✓ *IsValid:* Indica si todos los datos ingresados en los campos del formulario de edición son válidos, es decir cumplen con las condiciones dadas en la

especificación del caso de uso CUS-1 para los campos que componen el formulario. Esta variable hace parte de la validación para habilitar el botón de guardar, ya que si no es válida la interfaz, este botón no se activará.

- ✓ *IsDirty:* Indica si se han modificado los datos desplegados inicialmente por el formulario. Esta variable hace parte de la validación para habilitar el botón de guardar, ya que si no se ha modificado la interfaz, este botón no se activará.
- ✓ *ActiveWindow:* Indica el nombre de la ventana que está activa bajo una acción de usuario.
- ✓ *Items:* Indica la cantidad de empresas que componen la lista de la vista principal. Con esta variable se valida que luego de la acción de adicionar una empresa efectivamente la lista principal se modifique.

En la Figura D-20 se presenta una vista parcial (pues se generan más de 800 estados y transiciones) del modelo para este escenario, el cual se basa en el caso de uso CUS-1. Puede visualizarse el resultado consolidado del estado principal de la interfaz de acuerdo a las variables definidas anteriormente.

Figura D-20: Vista parcial de la máquina de estados generada para el escenario de adicionar una empresa nueva.


2"
ScenarioAddCompanyModel#0.CompanyViewScenario_1("1","AYAX LTDA.","Tomas Edisson","Cr 5 Av 4","2233445","2012-01-31")/"IsValid: True IsDirty: True ActiveWindow: Company Items : 2"
S9
ScenarioAddCompanyModel#0.Guardar_Company_1()/"IsValid: True IsDirty: False ActiveWindow: MsgProcessOK Items : 3"
S13
ScenarioAddCompanyModel#0.MsgAckProceesOk_Company_1()/"IsValid: True IsDirty: False ActiveWindow: Company Items : 3"
S17
ScenarioAddCompanyModel#0.Cancel_Company_1()/"IsValid: True IsDirty: False ActiveWindow: EmployersManager Items : 3"
S21
ScenarioAddCompanyModel#0.Close_1()/"IsValid: True IsDirty: False ActiveWindow: NotOpen Items : 3"
S25

D.4.2 Scenario: *UpdateCompanyModel*

Modela el comportamiento de la interfaz basándose en el caso de uso CUS-2 (Tabla D-5) y en los casos de prueba CP2-1 y CP2-2 (Tabla D-9). Se emulan las acciones que ejecuta un usuario para realizar la operación de modificar una empresa existente. Las acciones modeladas son:

- ✓ *LaunchView*: Emula la acción de abrir la interfaz Employers Manager.
- ✓ *SetSelectedItem*: Simula la acción de seleccionar un ítem de la lista de empresas de la vista principal. Tiene como parámetro de entrada el índice del ítem a seleccionar.
- ✓ *UpdateCompany*: Emula la acción de abrir el formulario de modificación de datos de una empresa existente.
- ✓ *CompanyViewScenario*: Emula la acción de usuario de ingresar los datos de edición de la empresa en los campos del formulario. Tiene como parámetros de entrada el Nombre, el Representante Legal, el Teléfono, la Dirección y la Fecha de Vinculación de la empresa.
- ✓ *Guardar*: Emula la acción de oprimir el botón de guardar del formulario.
- ✓ *MsgAckProceesOk*: Emula la respuesta de la interfaz indicando que el proceso de guardar la información se realizó correctamente.
- ✓ *Cancel*: Emula la acción de cancelar el formulario de modificación de datos.
- ✓ *Close*: Emula la acción de cerrar la interfaz principal.

Las variables que detallan el comportamiento de la interfaz para este escenario son:

- ✓ *HabilitarGuardar:* Indica si el botón de guardar debe estar habilitado. Se activará si todos los datos ingresados en los campos del formulario de edición son válidos (condiciones dadas en la especificación del caso de uso CUS-2) y si se han modificado los datos desplegados inicialmente por el formulario.
- ✓ *HablitarEdicion:* Indica si se debe activar el campo de Nit para edición.
- ✓ *ActiveWindow:* Indica el nombre de la ventana que está activa bajo una acción de usuario.
- ✓ *SelectedIndex:* Indica el índice del ítem seleccionado en la lista.

En la Figura D-3 se presenta una vista parcial del modelo para este escenario (pues se generan más de 400 estados y transiciones), el cual se basa en el caso de uso CUS-2. Puede visualizarse el resultado consolidado del estado principal de la interfaz de acuerdo a las variables definidas anteriormente.

Figura D-21: Vista parcial de la máquina de estados generada para el escenario de modificar los datos de una empresa existente.

D.4.3 Scenario: *DeleteCompanyModel*

Modela el comportamiento de la interfaz basándose en el caso de uso CUS-3 (Tabla D-6) y en los casos de prueba CP3-1 y CP3-2 (Tabla D-10). Se emulan las acciones que ejecuta un usuario para realizar la operación de eliminar una empresa. Las acciones modeladas son:

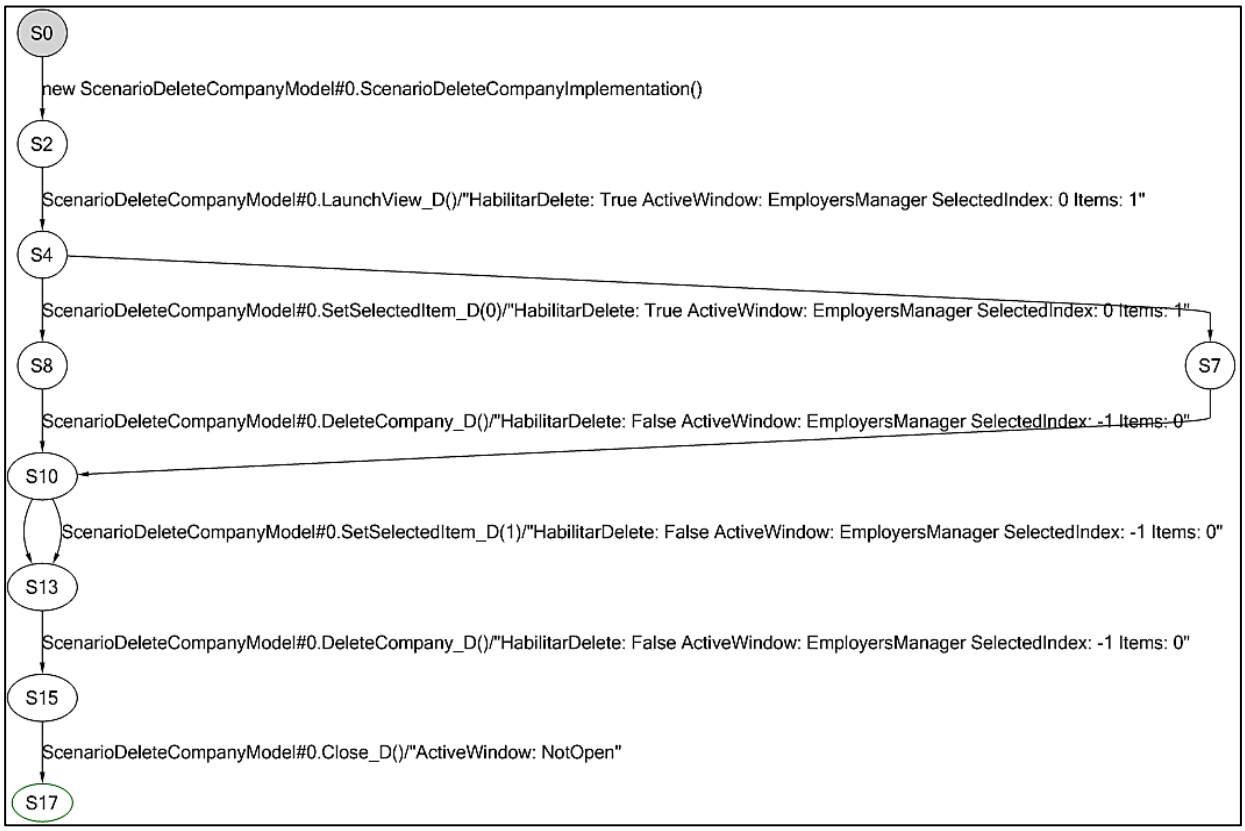
- ✓ *LaunchView*: Emula la acción de abrir la interfaz Employers Manager.
- ✓ *SetSelectedItem*: Simula la acción de seleccionar un ítem de la lista de empresas de la vista principal y el cual será eliminado. Tiene como parámetro de entrada el índice del ítem a seleccionar.
- ✓ *DeleteCompany*: Emula la acción de oprimir el botón de eliminar empresa.
- ✓ *Close*: Emula la acción de cerrar la interfaz principal.

Las variables que detallan el comportamiento de la interfaz para este escenario son:

- ✓ *HabilitarDelete:* Indica si el botón de eliminar del menú edición de la interfaz principal debe estar habilitado. Se activará si existe algún ítem en la lista principal (condición dada en la especificación del caso de uso CUS-3).
- ✓ *ActiveWindow:* Indica el nombre de la ventana que está activa bajo una acción de usuario.
- ✓ *SelectedIndex:* Indica el índice del ítem seleccionado en la lista.
- ✓ *Items:* Indica el número de items presentes en la lista de empresas.

En la Figura D-22 se presenta una vista parcial del modelo para este escenario (se generan más de 40 estados y transiciones), el cual se basa en el caso de uso CUS-3. Puede visualizarse el resultado consolidado del estado principal de la interfaz de acuerdo a las variables definidas anteriormente.

Figura D-22: Vista parcial de la máquina de estados generada para el escenario de eliminar una empresa.



D.4.3 Scenario: *FindCompanyModel*

Modela el comportamiento de la interfaz basándose en el caso de uso CUS-4 (Tabla D-7) y en los casos de prueba CP4-1 y CP4-2 (Tabla D-11). Se simulan las acciones que ejecuta un usuario para realizar la operación de buscar una empresa. Las acciones modeladas son:

- ✓ *LaunchView*: Emula la acción de abrir la interfaz Employers Manager.
- ✓ *SetSelectedItem*: Simula la acción de seleccionar un ítem de la lista de empresas de la vista principal. Tiene como parámetro de entrada el índice del ítem a seleccionar.
- ✓ *FindDialog*: Emula la acción de abrir el diálogo de búsqueda.
- ✓ *FindScenario*: Emula la acción de usuario de ingresar los parámetros de búsqueda en el diálogo. Tiene como parámetros de entrada el texto a buscar

(FindWhat), el campo por el cual buscar (Field) y si la búsqueda debe ser sensible a mayúsculas y minúsculas (MatchCase).

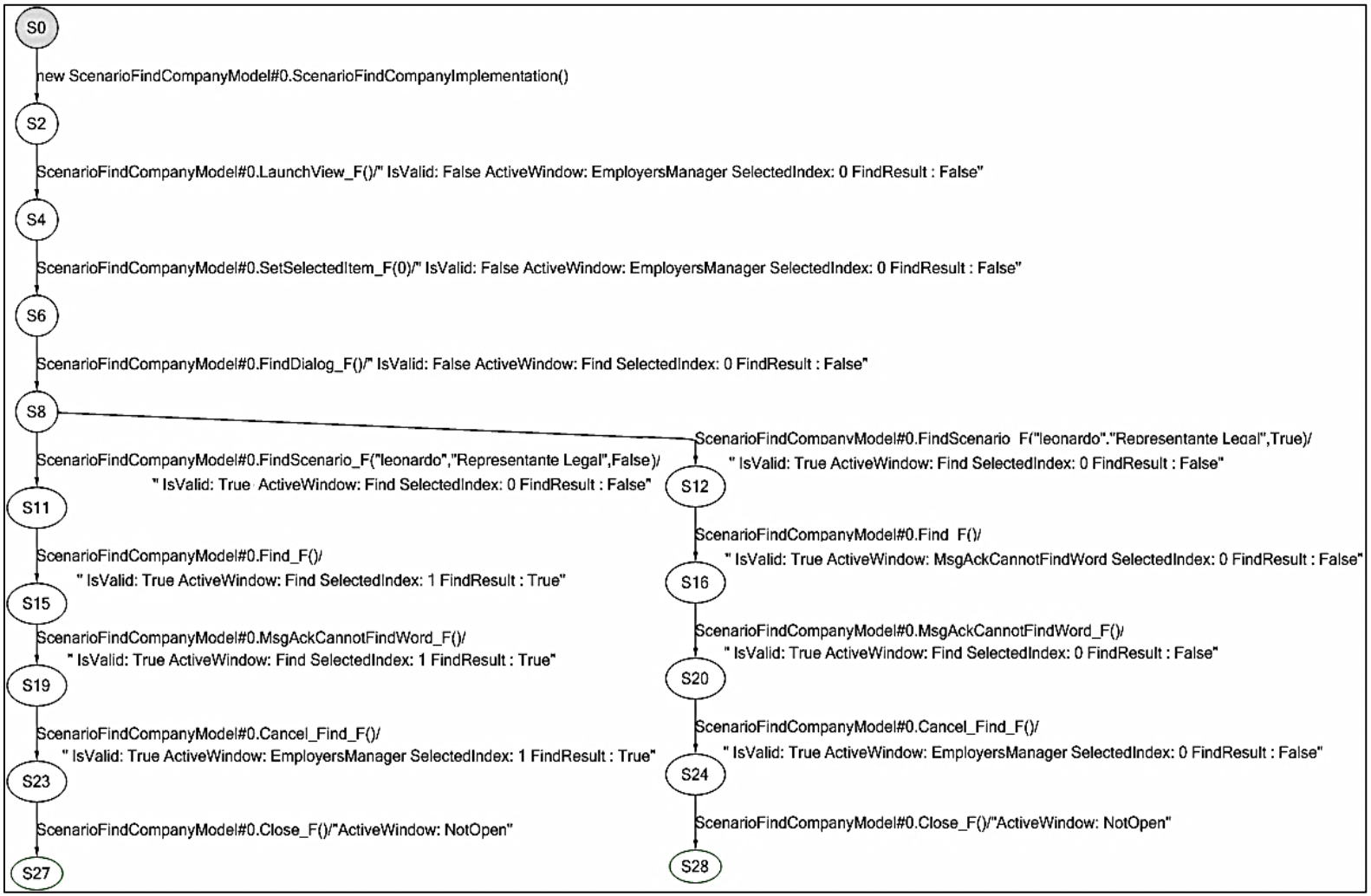
- ✓ *Find*: Emula la acción de oprimir el botón de Find en el diálogo.
- ✓ *MsgAckCannotFindWord*: Emula la respuesta de la interfaz indicando que no hubo resultados de la búsqueda.
- ✓ *Cancel*: Emula la acción de cancelar el diálogo.
- ✓ *Close*: Emula la acción de cerrar la interfaz principal.

Las variables que detallan el comportamiento de la interfaz para este escenario son:

- ✓ *IsValid:* Indica si todos los datos ingresados en los campos del diálogo son válidos, es decir cumplen con las condiciones dadas en la especificación del caso de uso CUS-4 para los campos que componen el diálogo.
- ✓ *ActiveWindow:* Indica el nombre de la ventana que está activa bajo una acción de usuario.
- ✓ *SelectedIndex:* Indica el índice del ítem seleccionado en la lista.
- ✓ *FindResult:* Indica si hubo o no resultados del proceso de búsqueda.

En la Figura D-23 se presenta una vista parcial del modelo para este escenario, el cual se basa en el caso de uso CUS-4. Puede visualizarse el resultado consolidado del estado principal de la interfaz de acuerdo a las variables definidas anteriormente.

Figura D-23: Vista parcial de la máquina de estados generada para el escenario de búsqueda de una empresa.



D.5 Generación de Pruebas

Para generar los casos de prueba a partir del modelo diseñado es necesario definir los valores de prueba, para ello se basó en la especificación en los escenarios de los casos de prueba CP1-1, CP1-2, CP2-1, CP2-2, CP3-1, CP3-2, CP4-1 y CP4-2 y en el criterio de cobertura definido: *Boundary value coverage* (ver sección 3.4).

En las tablas: Tabla D-12, Tabla D-13, Tabla D-14 y Tabla D-15 se presentan la descripción de los valores de prueba que se emplearán en las diferentes acciones que componen los escenarios de prueba modelados: El de agregar una nueva empresa, el de modificar una empresa existente, el de eliminar una empresa y el de buscar una empresa.

Definiendo los datos de entrada de pruebas en la herramienta se generan los casos de prueba de alto nivel para los escenarios especificados. En total se generaron automáticamente 376 casos de prueba de alto nivel con la combinación de todos los datos de entrada definidos para los diferentes contextos; para el escenario de agregar una nueva empresa se generaron 160, para el de modificar una empresa existente se generaron 96, para el de eliminar una empresa se generaron 12 y para el de buscar una empresa se crearon 108.

En las figuras: Figura D-24, Figura D-25, Figura D-26 y Figura D-27 se presentan ejemplos de los casos de prueba generados para cada escenario modelado. Se puede visualizar los parámetros de entrada de las acciones invocadas y los resultados esperados, el resultado de las acciones representa el comportamiento esperado de la interfaz en un punto determinado de ejecución.

Tabla D-12: Valores de prueba para el escenario de agregar una nueva empresa.

Acción	Parámetros de Entrada	Valores	Resultado Esperado
LaunchView	--	--	Se activa la interfaz principal.
AddNewCompany	--	--	Se activa el formulario de empresa.
CompanyView Scenario	Nit	"12345678", "999999999999999", "1000000000000000", "letras", "-1"	Solo se insertará la empresa cuando todos los valores ingresadossean válidos.
	Nombre	"AYAX LTDA.", " "	
	Representante Legal	"Tomas Edisson", " "	
	Teléfono	"2233445", " "	
	Dirección	"Cr 5 Av 4", " "	
	Fecha Vinculación	"2012-01-31", "2012-12-31"	
Guardar	--	--	Se actualiza la lista principal.
MsgAckProceesOk	--	--	Se activa un cuadro de mensaje informativo.
Cancel	--	--	Se desactiva el formulario de empresa.
Close	--	--	Se desactiva la interfaz principal.

Tabla D-13: Valores de prueba para el escenario de modificar una empresa existente.

Acción	Parámetros de Entrada	Valores	Resultado Esperado
LaunchView	--	--	Se activa la interfaz principal.
SetSelectedItem	lindex	-1,0,1,3	Se selecciona el item válido respectivo.
UpdateCompany	--	--	Se activa el formulario de empresa.
CompanyView Scenario	Nombre	"FENIX", "", "12345678901234567890123456789012345678901234567890123456789012"	Solo se modificará la empresa cuando todos los valores ingresados sean válidos.
	Representante Legal	"Campo Elias", "12345678901234567890123456789012345678901234567890123456789012"	
	Teléfono	7654321" , "1234567890123456"	
	Dirección	"Cr 6 Av 4", "12345678901234567890123456789012345678901234567890123456789012"	
	Fecha Vinculación	"2012-01-31", "2012-12-31"	
Guardar	--	--	Se actualiza la lista principal.
MsgAckProceesOk	--	--	Se activa un cuadro de mensaje informativo.
Cancel	--	--	Se desactiva el formulario de empresa.
Close	--	--	Se desactiva la interfaz principal.

Tabla D-14: Valores de prueba para el escenario eliminar una empresa.

Acción	Parámetros de Entrada	Valores	Resultado Esperado
LaunchView	--	--	Se activa la interfaz principal.
SetSelectedItem	lindex	-1,0,1,2,3,4	Se selecciona el item valido respectivo.
DeleteCompany	--	--	Se elimina de la lista el item seleccionado.
SetSelectedItem	lindex	-1,0,1,2,3,4	Se selecciona el item valido respectivo.
DeleteCompany	--	--	Se elimina de la lista el item seleccionado.
Close	--	--	Se desactiva la interfaz principal.

Tabla D-15: Valores de prueba para el escenario de buscar una empresa.

Acción	Parámetros de Entrada	Valores	Resultado Esperado
LaunchView	--	--	Se activa la interfaz principal.
SetSelectedItem	lindex	-1,0,1,3,4	Se selecciona el item válido respectivo.
FindDialog	--	--	Se activa el diálogo de búsqueda.
FindScenario	FindWhat	" ", "90909090", "leonardo", "Rollex", "XX", "12345678901234567890 12"	Solo se podrá realizar la búsqueda si todos los valores ingresados son válidos.
	Field	"Nit", "Nombre", "Representante Legal"	
	MatchCase	true, false	
Find	--	--	Si hubo resultados se selecciona el registro respectivo.
MsgAckCannot FindWord	--	--	Si no hubo resultados se activa un cuadro de mensaje informativo.
Cancel	--	--	Se desactiva el diálogo de búsqueda.
Close	--	--	Se desactiva la interfaz principal.

Figura D-24: Ejemplo de caso de prueba de alto nivel generado para el escenario de agregar una nueva empresa.

Figura D-25: Ejemplo de caso de prueba de alto nivel generado para el escenario de modificar una empresa existente.

Figura D-26: Ejemplo de caso de prueba de alto nivel generado para el escenario de eliminar una empresa.

Figura D-27: Ejemplo de caso de prueba de alto nivel generado para el escenario de buscar una empresa.

D.6 Implementación

El enfoque propuesto en el presente trabajo es modelar la interfaz aplicando el patrón MVP (secciones 3.3 y 4.2) .Con este modelo la vista de usuario es pasiva y es controlada totalmente por el presentador, de tal forma que validando el comportamiento del presentador se está implícitamente probando el comportamiento de la interfaz gráfica de usuario.

Una característica de este patrón es que existe un desacople entre el presentador y la vista, pues la interacción se realiza por medio de una interfaz y no se necesita saber los detalles de implementación de la GUI [29]. En la fase de modelado la vista simula el comportamiento deseado y en la fase de implementación la vista se concretiza con una tecnología determinada. Por esta razón, el diseño del modelo general de pruebas y el diseño de la interfaz concreta tienen la misma estructura, pero con diferentes detalles de implementación.

En la Figura D-28 se presenta un diagrama de dependencias con el diseño general del modelo de pruebas para los escenarios del caso de ejemplo. Estos modelos son diseñados con las facilidades que soporta *Spec Explorer 2010*.

En la Figura D-29 se presenta un diagrama de dependencias con el diseño de la interfaz gráfica de usuario concreta: *Employers Manager*, que tiene la misma estructura del modelo pero con los detalles reales de implementación. Si la interfaz esta acoplada con el modelo, quiere decir que cumple con los escenarios de los casos de uso definidos el sistema.

Como lo demuestran los diagramas se empleó la arquitectura propuesta para aplicar MBT sobre una vista compuesta (¡Error! No se encuentra el origen de la referencia. para l modelo de pruebas y ¡Error! No se encuentra el origen de la referencia. para la

implementación). La descripción general de los elementos principales que componen los diagramas del modelo (Figura D-28**)** y la implementación (Figura D-29) es[3]:

- ✓ ***ScenarioYYCompanyXX***: Abstraen los diferentes escenarios de pruebas que se definieron en los casos de uso.
- ✓ ***MainManagerEmployersXX***: Fachada principal para acceder al resto de funcionalidades.
- ✓ ***MainPresenterEmployersXX***: Es el componente fundamental, maneja el comportamiento de la vista principal (*MainViewEmployersCompanyXX*) y de los presentadores contenidos (*PresenterCompanyXX* y *PresenterFindXX*). La interacción entre este elemento y la vista es por medio de la interfaz *IMainViewEmployersCompany*.
- ✓ ***IMainViewEmployersCompany***: Expone las funcionalidades y atributos que debe tener la vista principal *Employers Manager* de acuerdo a los casos de uso. Se implementa en el modelo y en el componente concreto.
- ✓ ***MainViewEmployersCompanyXX***: Es la vista de usuario principal. En el modelo es un objeto que simula los eventos disparados por el usuario y en la implementación es el componente concreto diseñado con alguna tecnología. Implementa la interfaz *IMainViewEmployersCompany.*
- ✓ ***PresenterCompanyXX***: Maneja el comportamiento del formulario de edición de datos de empresa.
- ✓ ***PresenterFindXX****:* Maneja el comportamiento del diálogo de búsqueda.
- ✓ ***IViewCompany:*** Expone las funcionalidades y atributos que debe tener el formulario de edición de datos de empresa según los casos de uso.
- ✓ ***IViewFind:*** Expone las funcionalidades y atributos que debe tener el diálogo de búsqueda según los casos de uso.
- ✓ ***ViewCompanyXX:*** Es la vista que representa el formulario de edición de datos de empresa. En el modelo es un objeto que simula los eventos disparados por el usuario y en la implementación es el componente concreto. Implementa la interfaz *IViewCompany*.

[3] El símbolo XX indica que cambia el nombre del elemento dependiendo si es el modelo o la implementación, sin embargo tienen la misma responsabilidad dependiendo del contexto. El símbolo YY indica que cambia el nombre del elemento según el escenario.

- ✓ ***ViewFindXX:*** Es la vista que representa el diálogo de búsqueda. En el modelo es un objeto que simula los eventos disparados por el usuario y en la implementación es el componente concreto. Implementa la interfaz *IViewFind*. En el Modelado es el *ViewFindMBT* y en a implementación es el *FindDialog*.
- ✓ ***WindowManager:*** Utilidad que administra las ventanas, formularios, diálogos y cuadros de mensajes de la GUI. Se emplea en el modelo como en la implementación.

Figura D-28: Diseño general del modelo de pruebas para la interfaz gráfica de usuario *Employers Manager*.

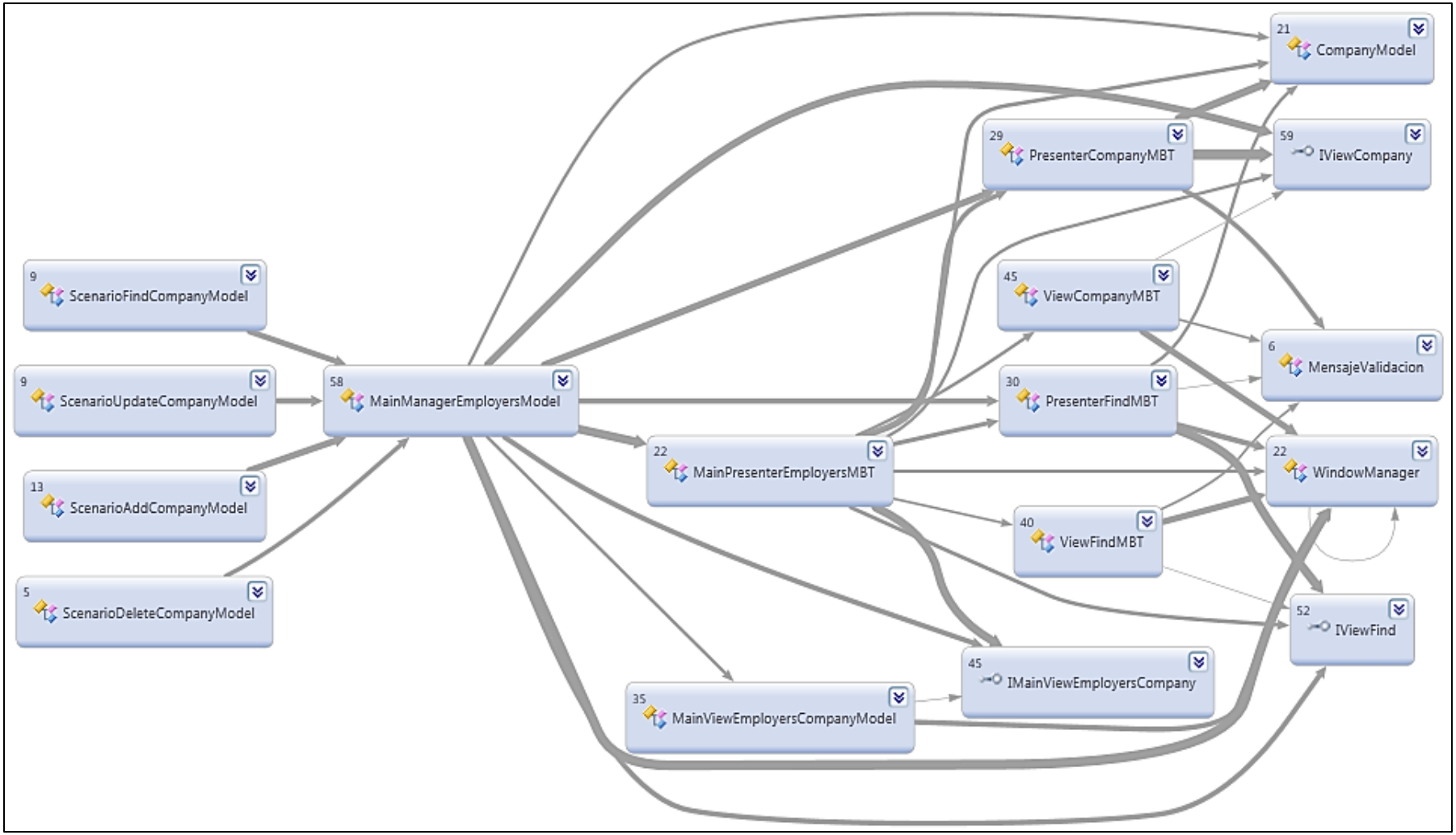
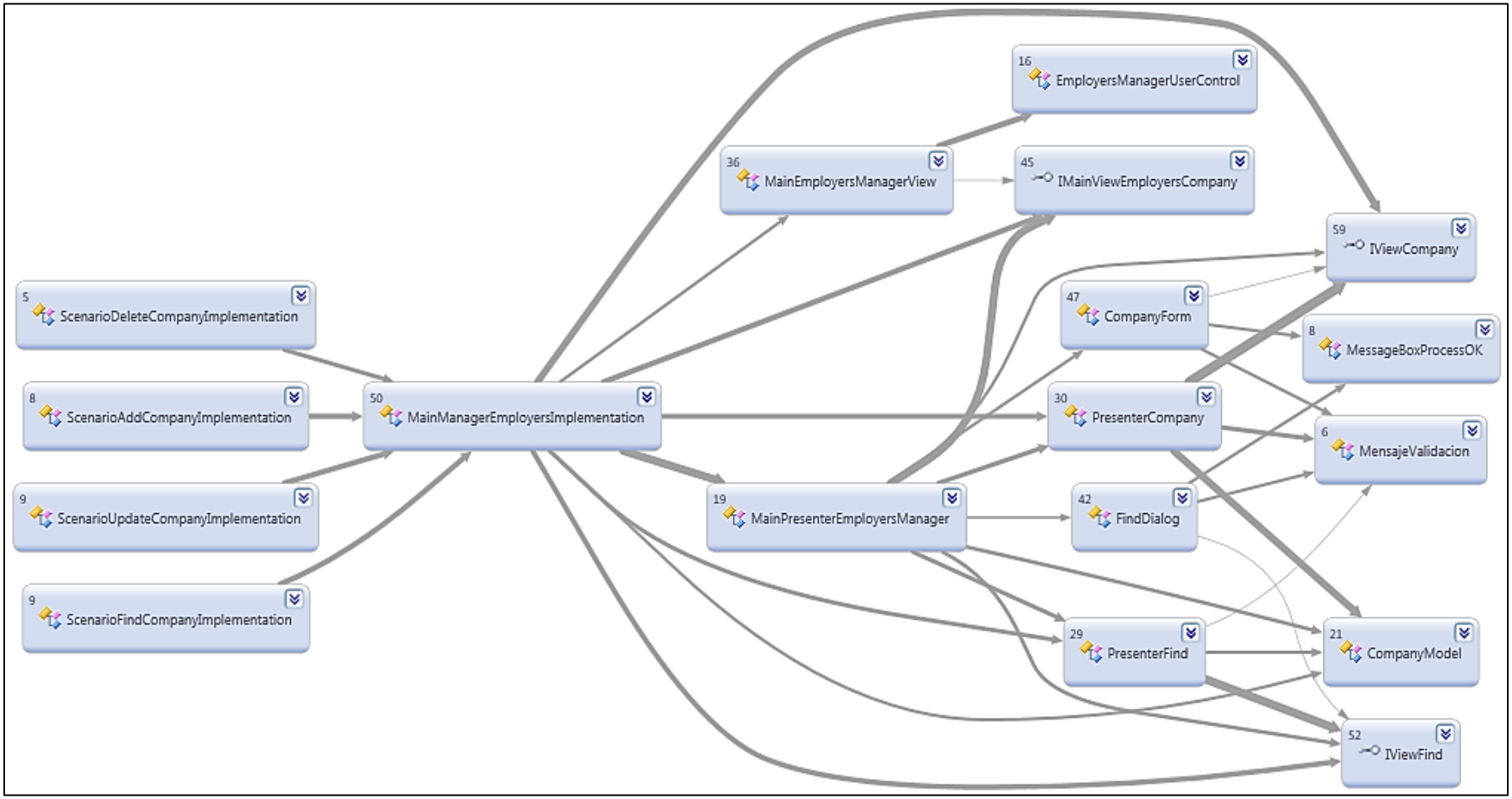


Figura D-29: Diseño general de la implementación concreta de la interfaz gráfica de usuario *Employers Manager*.



D.7 Resultados

Con la herramienta se generan los casos de pruebas ejecutables que apuntan directamente a la implementación. Finalmente se generaron 376 casos de pruebas ejecutables distribuidos en cuatro suites de pruebas (una por escenario):

- ✓ *ScenarioAddCompanyModelTestSuite:* Contiene 160 casos de pruebas para el escenario de agregar una empresa.
- ✓ *ScenarioUpdateModelTestSuite*: Contiene 96 casos de prueba para el escenario de modificar los datos de una empresa existente.
- ✓ *ScenarioDeleteModelTestSuite*: Contiene 12 casos de prueba para el escenario de eliminar una empresa existente
- ✓ *ScenarioFindModelTestSuite*: Contiene 108 casos de prueba para el escenario de buscar una empresa.

Luego de realizar tres iteraciones refinando los escenarios, el modelo y la implementación todas las pruebas pasaron y de esta manera se garantiza que la interfaz gráfica de usuario Employers Manager satisface los requerimientos definidos en los casos de uso CUS-1, CUS-2, CUS-3, CUS-4. A continuación se presentan algunas evidencias del comportamiento de la interfaz gráfica de usuario luego de acoplar la implementación con el modelo.

Figura D-30: Agregar una empresa nueva con datos inválidos.

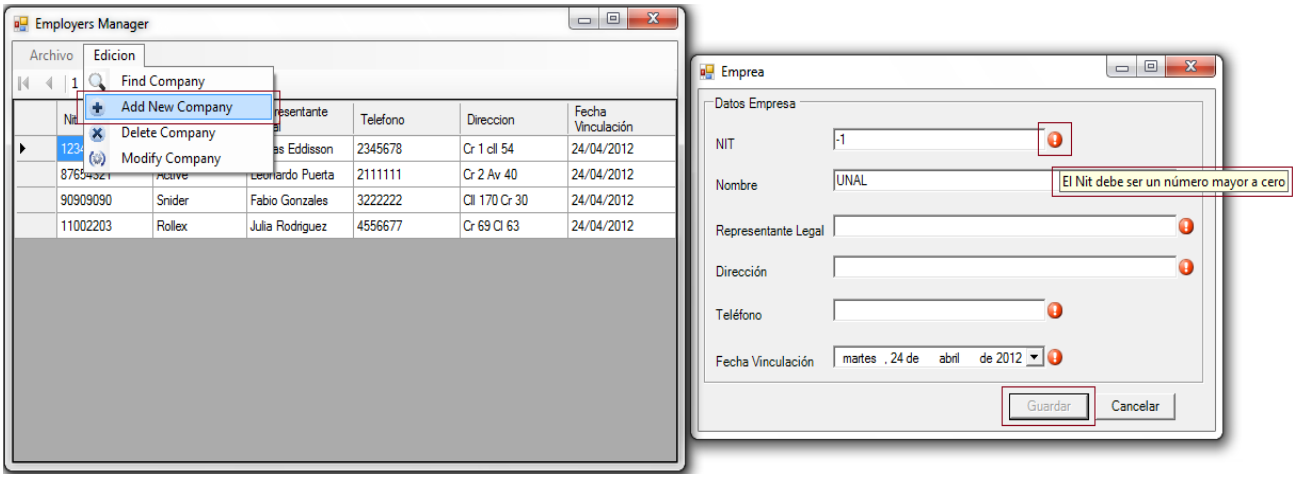


Figura D-31: Agregar una empresa nueva exitosamente.

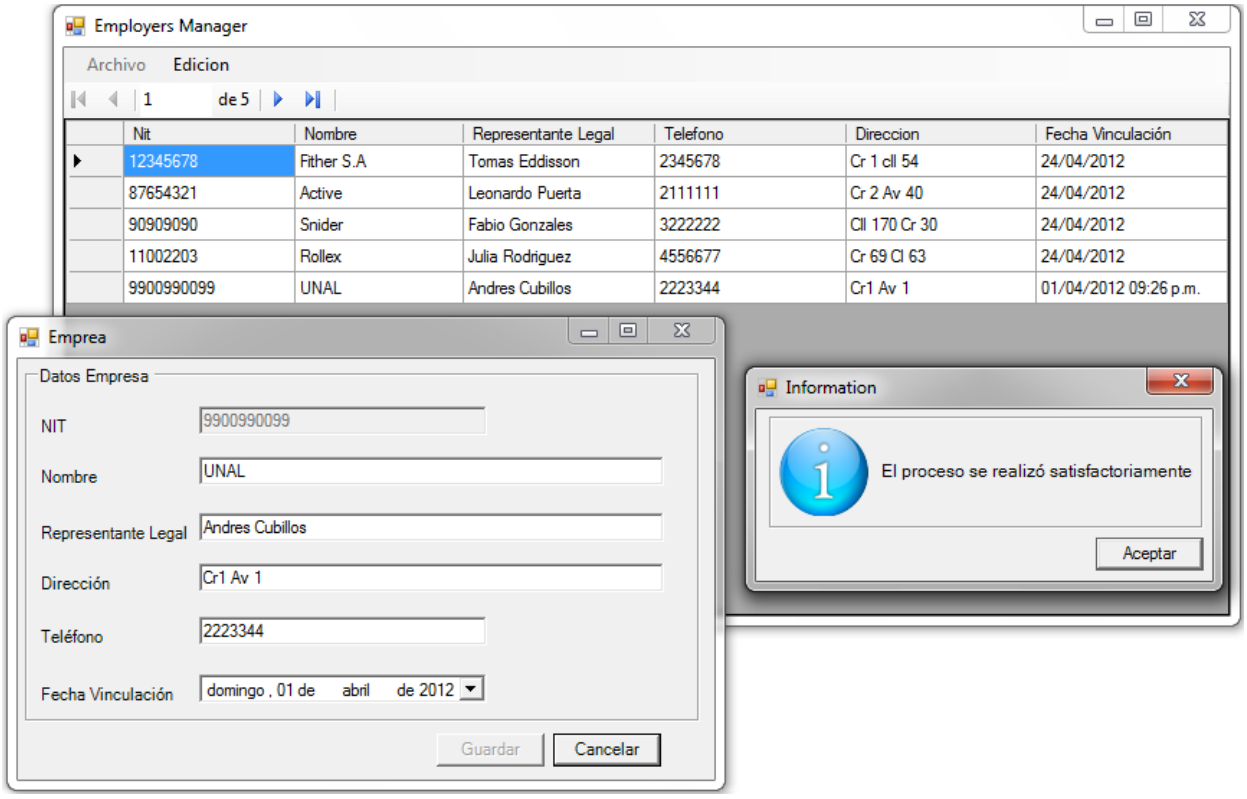


Figura D-32: Modificar una empresa con datos inválidos.

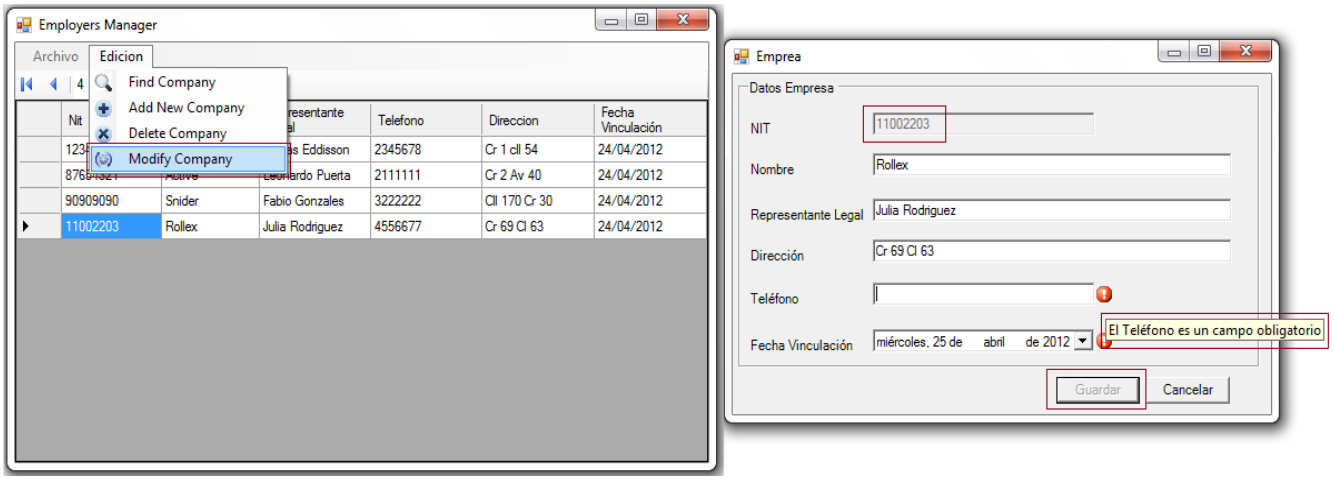


Figura D-33: Eliminar una empresa satisfactoriamente.

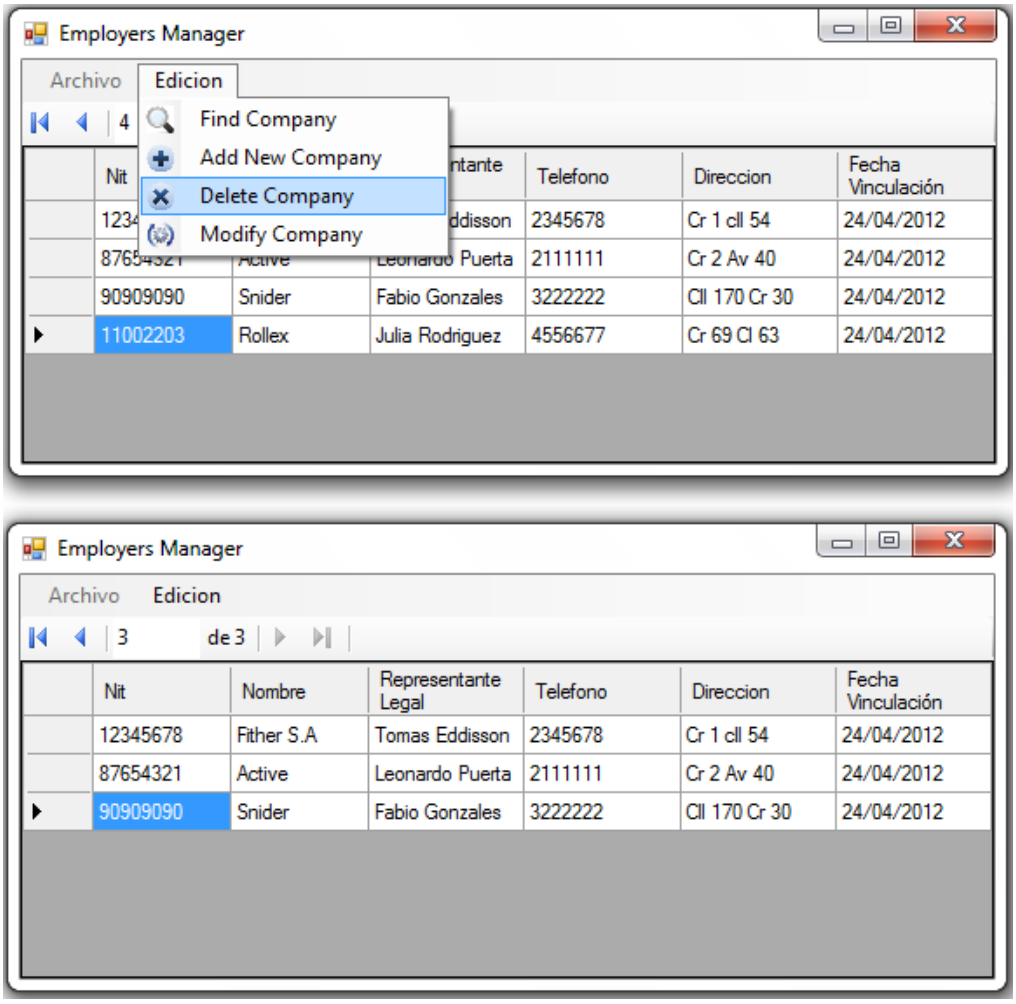


Figura D-34: Eliminar una empresa sin datos.

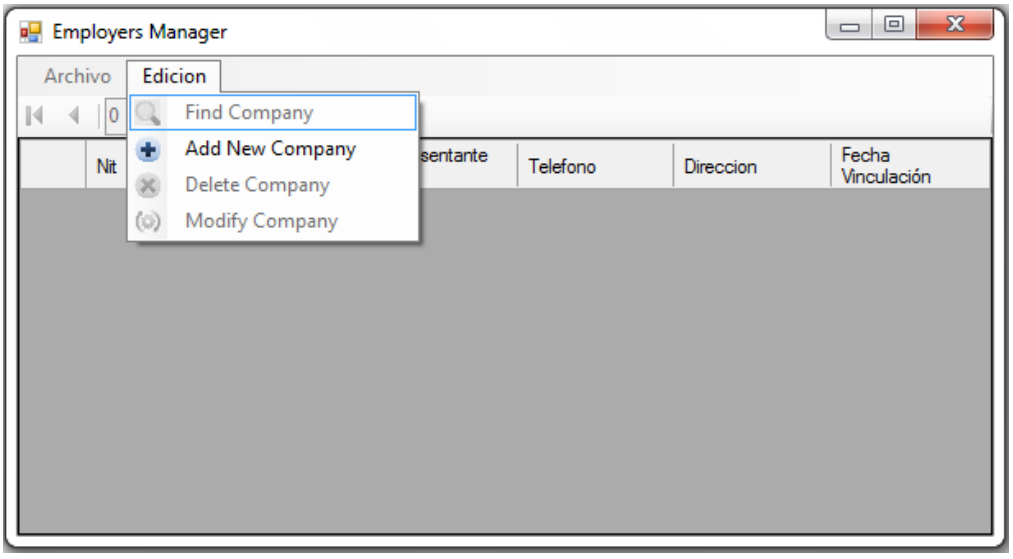


Figura D-35: Buscar una empresa exitosamente**.**

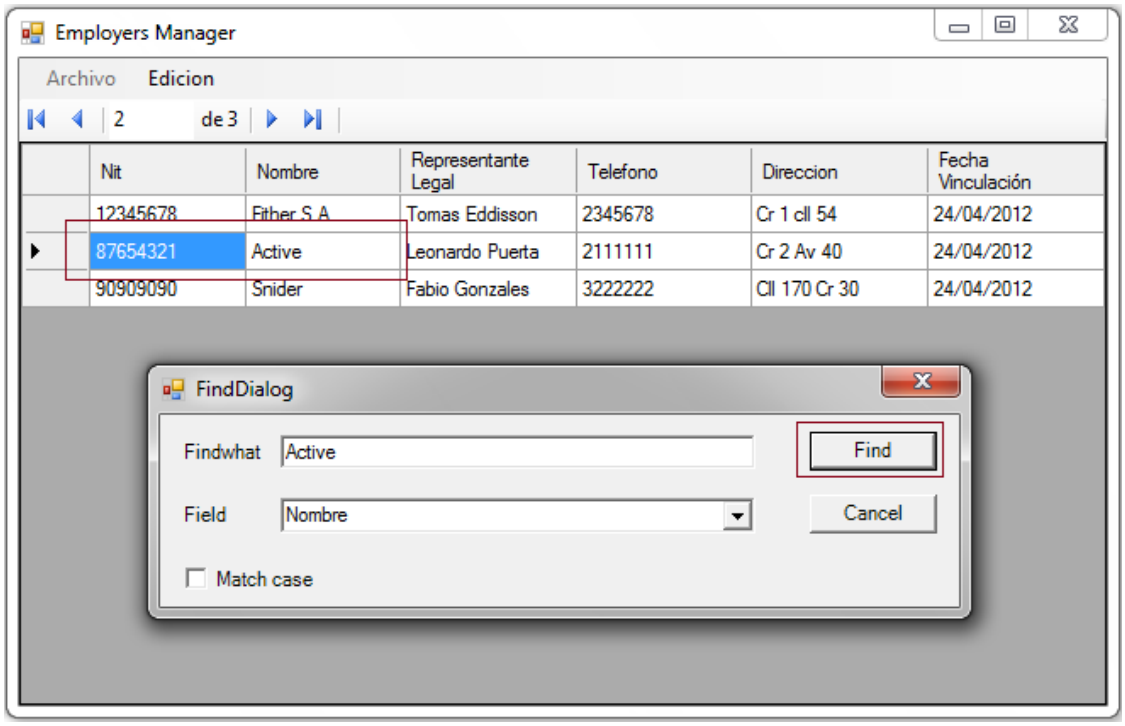


Figura D-36: Búsqueda sin resultados.

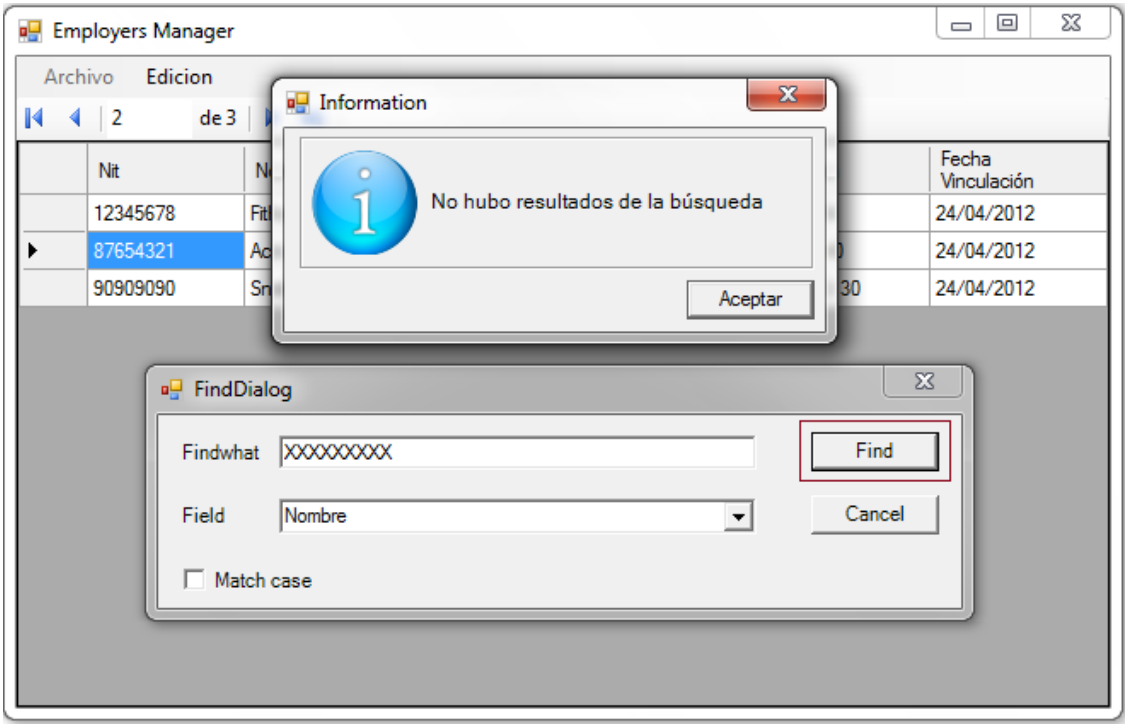
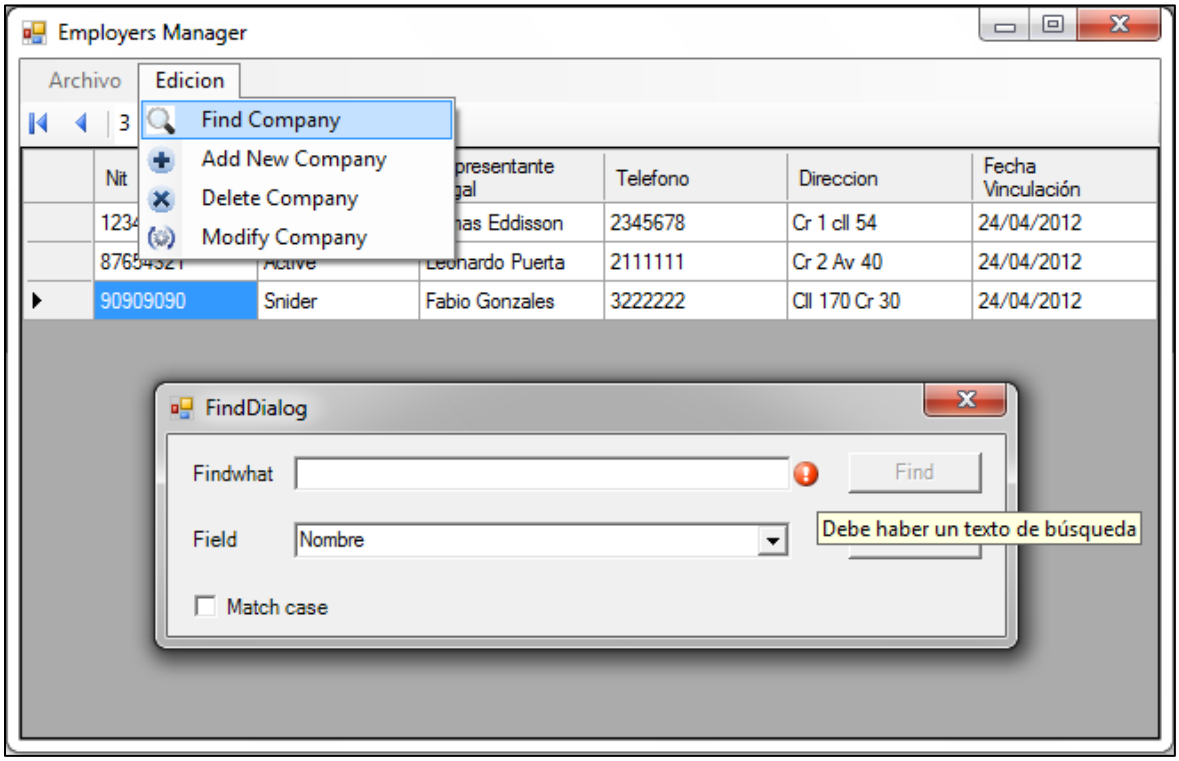


Figura D-37: Búsqueda con datos inválidos



Bibliografía

[1] UTTING, Mark, LEGEARD, Bruno. Practical Model Based Testing A Tools Approach. Morgan Kaufmann Title, 2007, 433p.

[2] PEREZ LAMANCHA. Beatriz. Pruebas Basadas En Modelos Para Líneas De Producto Software. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, Vol. 3, No. 4. Universidad de Castilla-La Mancha, Ciudad Real, España. 2009.

[3] NAVARRO, Pedro, MARTÍNEZ, Gregorio, SEVILLA, Diego. Aplicación de Open HMI Tester como Framework open-source para herramientas de pruebas de software. Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS), Vol. 5, No. 4, 2009.

[4] JORDÁN, C., ZÚÑIGA, A. Modelo Para Pruebas Automáticas De Software Aplicando Un Agente De Evolución Flexible. Escuela Superior Politécnica del Litoral. Revista Tecnológica ESPOL, Vol. 21, N. 1, Guayaquil - Ecuador. 2008, p. 99- 104.

[5] TORRES, Arturo, ESCALONA, María. Una Aproximación A Las Pruebas De Aplicaciones Web Basadas En Un Contexto MDWE. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, Vol. 3, No. 4. Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, 2009.

[6] NAVARRO, Pedro, MARTÍNEZ, Gregorio, SEVILLA, Diego. Verificación De Datos En La GUI Como Un Aspecto Separado De Las Aplicaciones. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, Vol. 4, 2010.

[7] NAVARRO, Pedro, MARTÍNEZ, Gregorio, SEVILLA, Diego. Proposal For Automatic Testing Of GUIs Based On Annotated Use Cases. Advances in Software Engineering 2010.

[8] WIECZOREK, S., STEFANESC, A. Improving Testing Of Enterprise Systems By Model-Based Testing On Graphical User Interfaces. Engineering of Computer Based Systems

(ECBS), 17th IEEE International Conference and Workshops. Marzo de 2010, p. 352 – 357.

[9] AYDAL, E., WOODCOCK, J. Automation of Model-Based Testing Through Model Transformations. Testing: Academic and Industrial Conference - Practice and Research Techniques. 2009, p. 63 – 71.

[10] ATIF, Memon. An Event-Flow Model Of GUI-Based Applications For Testing. Software Testing, Verification and Reliability. Volumen 17, No. 3, Septiembre de 2007, p. 137–157

[11] YONGZHONG, Lu, DANPING YAN, Chun Wang. Development Of An Improved GUI Automation Test System Based On Event-Flow Graph. Computer Science and Software Engineering, 2008 International Conference. Volumen 2. 2008, p. 712 – 715.

[12] PAIVA, Ana, FARIA, João. A Model To Implementation Mapping Tool For Automated Model-Based Gui Testing. Formal Methods and Software Engineering. Lecture Notes in Computer Science. Volumen 3785. 2005, p. 450-464.

[13] BEER, Armin, BANICA, Christian. Automated Software Testing. 2008.

[14] PAIVA, Ana, FARIA, João. Automated Specification-Based Testing Of Interactive Components With ASML. Proceedings of the 5th edition of the QUATIC international conference. Universidad de Porto. 2004.

[15] Microsoft Research. Spec Explorer. Disponible en: *Model-based Testing with SpecExplorer* .

[16] Microsoft Research. Abstract State Machine Language. Disponible en: *AsmL: Abstract State Machine Language*.

[17] BAKER, Paul, GRABOWSKI, Jens. Model-Driven Testing Using the UML Testing Profile. Springer. 2008, p. 183.

[18] FEWSTER, Mark, GRAHAM, Dorothy. Software Test Automation Effective Use of Test Execution Tools. Addison-Wesley. 1999, p. 570.

[19] JACKY, Jonathan, VEANES, Margus, CAMPBELL, Colin & SCHULTE, Wolfram. Model Based Software Testing and Analysis with C#. 2008, p. 349.

[20] WARMER, Jos & KLEPPE, Anneke. The Object Constraint Language - Getting Your Models Ready to MDA. 2003, p. 240.

[21] TORRES, Arturo, ESCALONA, María. Método de Pruebas De Sistema basado en Modelos Navegacionales en un Contexto MDWE. Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, 2009, p. 89.

[22] PEREZ LAMANCHA. Beatriz. Propuesta para pruebas dirigidas por modelos usando el perfil de pruebas de UML 2.0. Universidad de la República de Uruguay .Revista Española de Innovación, Calidad e Ingeniería de Software (RECIS). 2008.

[23] DUAN, Linshu, HOFER, Alexander & HUSSMANN, Heinrich. Model-Based Testing Of Infotainment Systems On The Basis of a Graphical Human-Machine Interface. Second International Conference on Advances in System Testing and Validation Lifecycle. 2010, p. 5-9.

[24] PAIVA, Ana Cristina. Automated Specification – Based Testing of Graphical User Interfaces. PhD Tesis. Departamento de Ingeniería Eléctrica y Computación. Universidad de Porto. 2006, p. 252.

[25] PAU, Valentin, MIHAILESCU, Marius, STANESCU, Octavia. Model View Presenter Design Pattern. Journal of Computer Science and Control Systems. Vol. 3, no.1. 2010, p 173 - 176.

[26] SHAFIQUE, Muhammad, LABICHE Yvan. A Systematic Review of Model Based Testing Tool Support. Software Quality Engineering Laboratory, Department of Systems and Computer Engineering, Carleton University. 2010, p. 21.

[27] SOMMERVILLE, Ian. Ingeniería de Software. Sexta edición. Pearson Educación. 2002, p. 692.

[28] PRESSMAN, Roger. Ingeniería del software, un enfoque práctico. Sexta edición. McGraw-Hill Interamericana. 2006, p. 958.

[29] ZHANG, Yang, LUO, Yanjing. An Architecture and Implement Model for Model-View-Presenter Pattern. Computer Science and Information Technology (ICCSIT), 3rd IEEE International Conference. 2010, p. 532- 536.

[30] ALLES, Micah, CROSBY, David. Presenter First: Organizing Complex GUI Applications for Test-Driven Development. Proceedings of AGILE 2006 Conference (IEEE). 2006, p. 10.

[31] MISCKEI, Zoltan. Model-based software testing. Disponible en la web page del autor: Model-based software testing web page.

[32] ROBINSON, Harry. Model-Based Testing Home Page. Disponile en la web page: Model-Based Testing Home Page.

[33] WEYUKER, Elaine. Can we measure software testing effectiveness. Software Metrics Symposium Proceedings. First International Carleton University. 1993, p. 100 - 107.