Capitulo

Alternativas metodológicas y técnicas para la generación de pruebas a partir de modelos.

HENRY ROBERTO UMAÑA A., Profesor Asociado, hrumana@unal.edu.co
MIGUEL ANGEL CUBIDES G., Investigador, macubidesgo@unal.edu.co

*ColSWE, **Col**ectivo de investigación en ingeniería de **S**oft**W**ar**E**, Ingeniería de Sistemas e Industrial. Universidad Nacional de Colombia.*

Resumen

*Este trabajo presenta tres aproximaciones metodológicas y técnicas para la generación de pruebas a partir de modelos: SCENT (Method for **SCEN**ario-Based Validation and **T**est of Software), UML-Based Statistical Test Case Generation y AGEDIS (**A**utomated **G**eneration and **E**xecution of Test Suites for **DI**stributed Component-Based **S**oftware). Las dos primeras plantean una aproximación metodológica y la utilización de estereotipos basados en artefactos estándar de UML y la última ofrece además un conjunto de herramientas para la automatización total del proceso.*

Abstract

*This paper introduces three methodological and technical approach for models based test generation: SCENT (Method for **SCEN**ario-Based Validation and **T**est of Software), UML-Based Statistical Test Case Generation and AGEDIS (**A**utomated **G**eneration and **E**xecution of Test Suites for **DI**stributed Component-Based **S**oftware). The first two use stereotypes based in UML standard artefacts and the last one offers as well as a complete set of tools for the total automation of the testing process.*

1. Introducción

Los procesos de verificación y validación del software consumen una gran cantidad del esfuerzo total (tiempo, actividades, recursos) de la producción del mismo. Tradicionalmente se definen los casos de prueba y los datos de prueba para las especificaciones funcionales y no funcionales de una manera formal pero manual, es decir son los encargados de control de calidad los que especifican, ejecutan y comunican los resultados de la verificación y de la validación. Una ganancia en el proceso de verificación consistiría en avanzar en la sistematización y si se puede en la automatización del mismo.

En este capítulo se sintetizan tres metodologías que proponen pruebas basándose en modelos: SCENT, UML-Based Statistical Test Case Generation y AGEDIS.

La organización de este capítulo es la siguiente: en la Sección 2 especificamos brevemente el marco teórico. A continuación en la sección 3 presentamos las tres metodologías referenciadas. De cada una de ellas se presenta una visión general, el procedimiento para la obtención de pruebas y un ejemplo que muestra la aplicación de la metodología ó en el caso de AGEDIS, de las herramientas que permiten su implementación. Finalmente, en las últimas secciones las conclusiones de esta revisión teórica, hacia dónde dirigiremos nuestros esfuerzos, los agradecimientos y las referencias bibliográficas.

2. Contenido

Model-based Testing es una técnica en la cual los casos de prueba son generados total o parcialmente de los modelos que describen ya sea la estructura ó el comportamiento del sistema a probar. Los casos de prueba pueden ser generados manualmente pero existen iniciativas tendientes a automatizar este proceso.

Los casos de prueba generados por medio de metodologías y técnicas tendientes a la automatización pueden ser ejecutados en línea ó fuera de línea. En línea significa que la herramienta que genera los casos de prueba se conecta directamente con el sistema y ejecuta las pruebas proveyendo los resultados. Fuera de

línea significa que los casos de prueba generados ó bien constituyen algún conjunto de instrucciones que el sistema entiende y que de alguna manera se acopla y se puede ejecutar después ó bien son un conjunto de guías para construirlos y posteriormente ejecutarlos.

3. Alternativas metodológicas y técnicas para la generación de pruebas

Una de las formas de avanzar en la automatización de pruebas es la de visualizar el sistema como una máquina de estado finito modelada con diagramas de transición de estados. En algunos casos se derivan a partir de la especificación ampliada de escenarios de los casos de uso (SCENT y UML-Based Statistical) ó a partir del diagrama de clases (AGEDIS). Todas las posibles rutas se convierten en los casos de prueba. Una manera de reducir el gran número de combinaciones posibles es colocarle peso a las posibles rutas por su probable uso basado en determinadas estadísticas (UML-Based Statistical).

3.1. SCENT

SCENT es una metodología que propone ampliar la idea de analizar escenarios para la especificación y elección de requerimientos hacia una visión mucho más completa que abarca la generación de pruebas.
A pesar de que la visión de aprovechar los escenarios para la generación de casos de prueba ha sido trabajada desde ya hace algún tiempo, fue a partir de SCENT que se definió la metodología y proceso a seguir para obtener, a partir de estos escenarios, una colección de pruebas. Sin embargo, los escenarios al ser desarrollados en lenguaje natural, no son fácilmente utilizables para automatizar la generación de pruebas, es por esto que SCENT ha ampliado los escenarios introduciendo nuevas características como anotaciones especiales o un diagrama de dependencias de ellos.

3.1.1. Visión General. El método SCENT (Method for **SCEN**ario-Based Validation and **T**est of Software[1] propone desarrollar un conjunto de pruebas para el software a partir de los escenarios, con lo que se aprovechan dos factores, a saber, la integración de la generación de pruebas desde las primeras fases del desarrollo de la aplicación, y, por otra parte pero ligada a ésta, el aprovechamiento de los resultados obtenidos en las fases de análisis en futuras fases, específicamente en la de generación e implementación de pruebas.
SCENT también permite aprovechar los escenarios (además de documentar y describir la funcionalidad y el comportamiento del sistema) para validar el sistema desde la fase de análisis, de manera que el sistema finalmente desarrollado podrá ser más robusto, al procesar un análisis más preciso y específico, todo esto bajo la percepción de los actuales paradigmas de desarrollo que (desde el orientado a objetos) manejan una especificación de escenarios, que podrían ser ampliados con la especificación SCENT para obtener a partir de ellos un conjunto de pruebas.
SCENT implementa escenarios resolviendo ciertos inconvenientes que estos presentan en su implementación para generar casos de prueba [2]. Dichos inconvenientes son:

- Los escenarios no son formales: Usualmente se maneja lenguaje natural para definir los escenarios y debido a esto no es posible generar pruebas automáticamente a partir de ellos. Para solventar este problema, SCENT utiliza diagramas de estados para formalizar los escenarios.
- Los escenarios son utilizados en la fase de requerimientos solamente: Los escenarios son utilizados para definir y especificar requerimientos solamente. Sin embargo, SCENT propone utilizarlos para generar los casos de prueba y además como un soporte que posiblemente será aprovechado en cualquier otra fase de desarrollo.
- Los escenarios describen a groso modo el sistema desde un nivel abstracto: A pesar de que los escenarios exhiben el sistema desde un punto de vista de usuario (desde una perspectiva dinámica), las pruebas generadas automáticamente a partir de ellos (a pesar de no ser estáticas) abarcarían el software completo y serían de gran ayuda.

3.1.2. Procedimiento para la obtención de pruebas. La metodología SCENT propone seguir diferentes pasos que clasifican en tres super categorías [1]:

3.1.2.1. Creación de escenarios. La creación de los escenarios empieza por la definición de los actores especificando sus roles, seguidamente se definen los escenarios empezando por una manera somera y luego perfeccionándolos ingresando más información como flujo de acciones, excepciones, errores, manejo de excepciones e información referente a requerimientos no funcionales. A partir de estos escenarios se generan diagramas con los que se obtendría una retroalimentación con el usuario final.
Los diagramas de dependencias son introducidos para capturar dependencias lógicas y temporales entre estados. SCENT define tres tipos de dependencias diferentes: de abstracción, temporales o causales. Para la notación se dibuja un rectángulo redondeado con dos círculos en los extremos para definir escenarios y

conexiones entre los círculos para definir dependencias.
Una descripción más profunda de este proceso se puede encontrar en [2]. Sin embargo, algunas de estas tareas pueden ser ejecutadas paralelamente, e, incluso, en un orden diferente. SCENT especifica que esta relación no es lineal, de manera que efectivamente se puede escoger cualquier camino. La idea es tener los escenarios validados y bien documentados para el siguiente paso.

3.1.2.2. Formalización de los escenarios. El proceso de formalización de los escenarios permite generar diagramas de estados a partir de los escenarios. Debido a que los escenarios son descritos en lenguaje natural, en este proceso se filtran posibles errores, inconsistencias y omisiones, y, debido a esto, la formalización de los escenarios es un proceso que debe realizarse iterativamente junto a la de creación de escenarios.
La información obtenida en el paso anterior debe ser documentada diligenciando el formato que SCENT presenta para este fin.
A pesar que SCENT no define un procedimiento para la generación de los diagramas de estados, si define algunas reglas:

- Se debe crear un diagrama de estados por cada escenario (definiendo todos los flujos de ejecución posibles).
- Generalmente cada paso simple en un escenario se convierte en un estado o transición en un diagrama de estados.
- Se sugiere modelar el flujo principal primero, luego los alternativos.
- Los escenarios abstractos se convierten en diagramas padres con sentido de herencia en los diagramas de estados.
- Revisar que los diagramas de estados sean consecuentes con los escenarios y tengan sentido en si mismos.

Por otra parte, SCENT introduce información adicional a los diagramas de estados. Esta información es:

- Precondiciones
- Entradas, salidas y rangos de datos
- Requerimientos no funcionales

3.1.2.3. Derivación de los casos de prueba. Para la derivación de los casos de prueba se llevan a cabo tres pasos:

- Derivación de casos de prueba a partir de diagramas de estados: Para este paso se puede utilizar cualquier método para cruzar todos los caminos posibles del diagrama de estados ó cualquier otro método de manera que se pueda optimizar la cantidad de pruebas realizadas.
- Derivación de casos de prueba a partir de diagramas de dependencias: Todas las dependencias deberán generar casos de prueba diferentes. Por ejemplo, si un escenario debe ser precedido por otro, debe probarse lo que sucedería si es o no precedido por este.
- Pruebas adicionales: Estas pruebas son generadas a partir de anotaciones adicionales que se hayan realizado durante la fase de análisis (éstas resultan en la fase de formalización de escenarios)

3.1.3. Ejemplo. En la documentación de SCENT [1] se propone un caso de un ATM definido como sigue:

1. El cliente ingresa la tarjeta
2. El sistema verifica la validez de la tarjeta
3. El sistema muestra el diálogo: ingresar contraseña
4. El cliente ingresa la contraseña
5. El sistema verifica la contraseña
6. El sistema muestra el menú principal

Los diagramas serían:

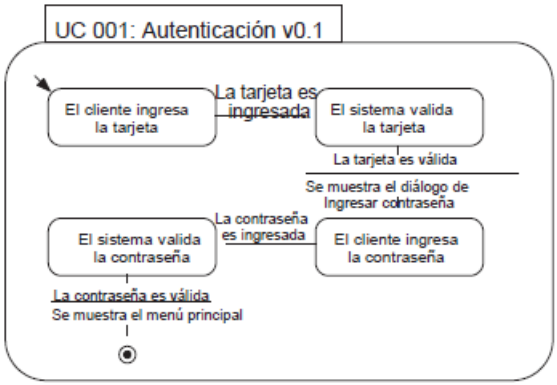

Figura 1. Diagrama de estados representando el escenario "Autenticación"

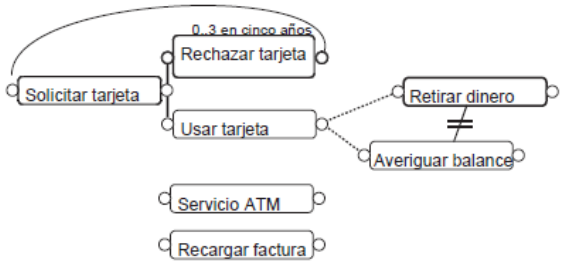

Figura 2. Diagrama de dependencias

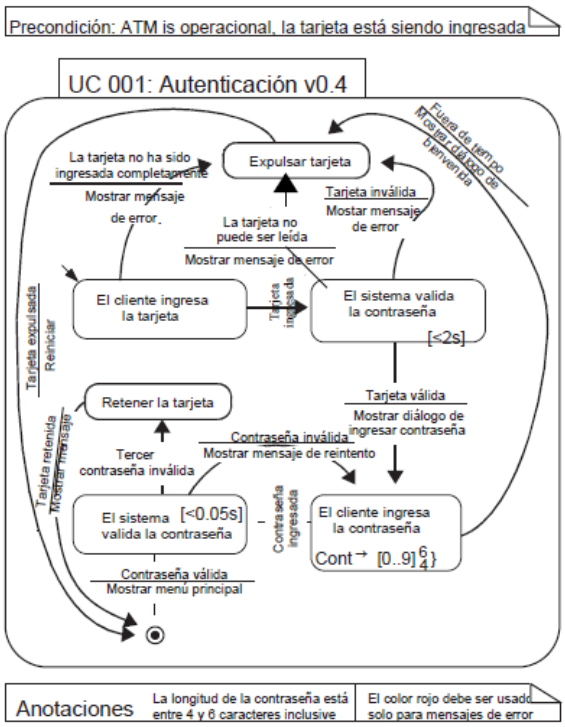

Figura 3. Diagrama de estados extendido

3.2. UML-Based Statistical Test Case Generation

Esta es una metodología que propone generar casos de prueba que abarquen el sistema desde un punto de vista estadístico, soportándose en la idea de que tan solo una mínima parte de código es ejecutada el mayor tiempo. Con este fin, se debe analizar previamente la ocurrencia de ejecución de cada segmento de la aplicación.
Para lograr este objetivo, se propone aprovechar en primera medida diagramas UML, específicamente los diagramas de casos de uso y los diagramas de estado. Aprovechando los diagramas de casos de uso y definiendo previamente la ocurrencia de cada caso de uso, se espera lograr un conjunto de pruebas de carácter dinámico, mientras que con los diagramas de estado el objetivo es generar pruebas de carácter estático.

3.2.1. Visión General. UML-Based Statistical Test Case Generation[5] es una metodología que propone generar casos de prueba a partir de la fase de análisis, enfocándose en obtener pruebas con parámetros de probabilidad de ocurrencia sobre cada segmento del programa.
Esta metodología se basa en la idea de generación de pruebas sobre uso estadístico, de manera que un segmento del programa que sea ejecutado en mayor escala que otro, será más cuidadosamente probado. Para poder conocer qué segmentos del programa son ejecutados más frecuentemente, se aprovechan los modelos de uso que se obtiene un esquema de ejecución del software final [6].
Para obtener una automatización en la generación de pruebas, esta metodología propone iniciar su desarrollo a partir de diagramas de casos de uso y diagramas de estado con la especificación UML. Para esto los casos de uso deberán ser ampliados introduciendo, por ejemplo, información de ocurrencia por cada caso de uso.

3.2.2. Procedimiento para la obtención de pruebas. Esta metodología propone seguir cinco pasos para obtener los casos de prueba, estos son:

3.2.2.1. Refinado de casos de uso. Los casos de uso deben ser enfocados desde una perspectiva de usuario, deberán contener precondiciones, post condiciones, los escenarios de ejecución principal y uno o varios alternativos, escenarios de respuestas a situaciones excepcionales (por ejemplo ingreso de datos inválidos) y los casos de uso incluidos. Para generar la documentación de los casos de uso, esta metodología propone un formato[5].

3.2.2.2. Generación de diagramas de estado a partir de los casos de uso. Debido a que los casos de uso contienen otros casos de uso incluidos, se sugiere generar un diagrama de estado de cada uno de ellos, y otro general que presente la navegabilidad. Cada diagrama deberá contener información de todos los diferentes escenarios de cada caso de uso, y, en el caso del diagrama general se incluirá además información del evento que genera la ejecución de un caso de uso diferente. Para graficar los diagramas de cada caso de uso se debe seguir:

- Generar un estado para cada pre y post condición del caso de uso. Estos estados deben contener información de la condición a la que se refieren.
- Se generan los estados del caso de uso. Estos deberán ser graficados de manera que solo contengan el nombre, y, de tenerlo, un estado anidado de cada flujo alternativo.
- En los estados propios del caso de uso que estén conectados a un estado de pre o post condición, estarán representando el flujo principal
- Para representar el flujo alternativo se deberá partir del estado anidado que lo represente y llegar al estado al que corresponda.
- Finalmente se agregan las transiciones de inicio y finalización del sistema.

Y para el caso de uso general se debe seguir:

- Cada caso de uso será representado por un estado.
- Tanto los estados como las relaciones deberán ser nombradas para mayor entendimiento del flujo de cada escenario.
- Las post condiciones se convierten en conectores de manera que un estado puede tener más de un conector debidamente nombrado que se refiere a una ejecución diferente.
- Las precondiciones no deberán ser graficadas, pues se considera que están modeladas en el diagrama referente al caso de uso particular.

3.2.2.3. Generación de diagramas de uso a partir de diagrama de estados. En este paso se deberá crear un diagrama de uso por cada diagrama de estado general. Un diagrama de uso es básicamente un diagrama de estados extendido, de manera que se le agrega un nombre que califica a la acción que el usuario ejecuta para llegar al nuevo estado. Para lograr esto se deben seguir los pasos:

- Se debe sustituir recursivamente todos los estados por el diagrama al que representan
- Los estímulos deben unificarse, de manera que si se repite un mismo estímulo deberá ser nombrado de igual manera
- Las transiciones automáticas entre estados deberán ser etiquetadas con la letra *Epsilon*.
- La transición inicial deberá ser nombrada "Software Not Invoked"
- La transición final deberá ser nombrada "Software Terminated"
- No debe existir ningún estado final ni estados inalcanzables.

3.2.2.4. Generación de modelos de uso a partir de diagramas de uso. En este paso se deberá agregar una etiqueta con la probabilidad de ocurrencia de cada transición. El caso particular de las transiciones denotadas como *Epsilon* la probabilidad de ocurrencia sería 1, al igual que en el caso de la referente a "Software Not Invoked". Se entiende que el siguiente estado al de "Software Terminated" es "Software Not Invoked". Aprovechando este hecho se puede afirmar que todos los estados tendrán un siguiente estado. El cálculo de esta probabilidad no está especificado por la metodología.

3.2.2.4. Hacia los casos de prueba a partir de modelos de uso. Para generar los casos de prueba se siguen caminos de manera aleatoria aprovechando la información de la probabilidad de ocurrencia, de manera que entre mayor probabilidad tenga, más probable será la generación de una prueba por ese camino. La ejecución de una prueba podrá terminar como correcta o con falla. En caso de haber una falla se deberá revisar esta falla y calificarla como de alta o baja relevancia y se deberá decidir si se continúan las pruebas o se detienen.

3.2.3. Ejemplo. En la documentación de UML-Based Statistical Test Case Generation[5] se propone un ejemplo referente al préstamo de un libro en una biblioteca así:

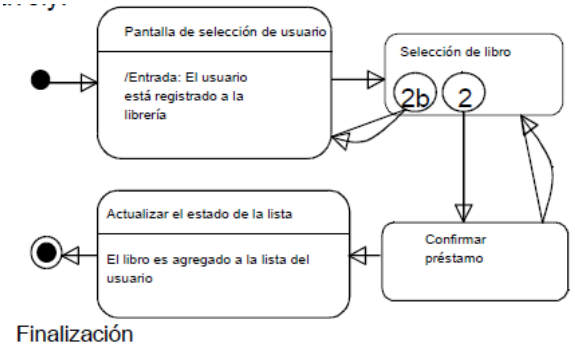

Figura 4. Diagrama general del caso de uso prestar libro

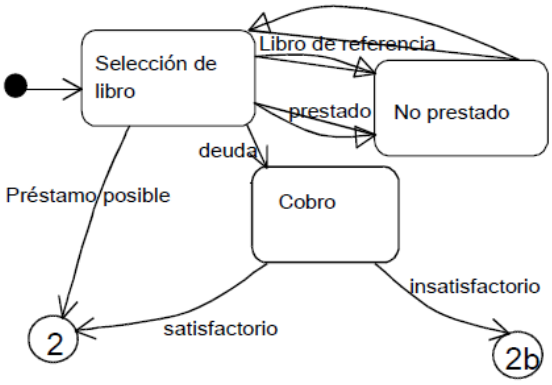

Figura 5. Diagrama refinado de caso del uso prestar libro

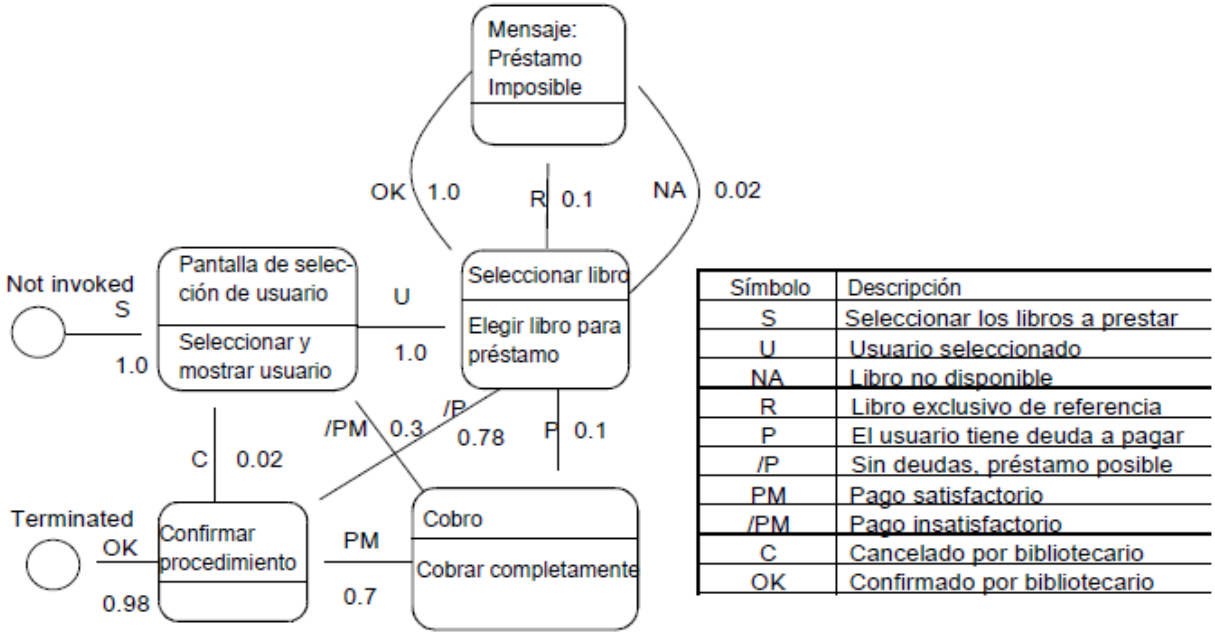

Figura 6. Modelo de uso del caso de uso prestar libro

3.3. AGEDIS

AGEDIS es una metodología para la generación y ejecución automática de pruebas a partir de modelos, promovida por la Comisión Europea. Este proyecto que inició en el año 2000 tuvo como objetivo el desarrollo de una metodología soportada con herramientas para su automatización, y se invirtieron 4.3 billones de euros para su consecución.
Para lograr una automatización en la generación de pruebas, AGEDIS propone el uso de diagramas con especificación UML con ciertas anotaciones particulares propias de la metodología que definen al sistema a desarrollar junto a una serie de directivas de generación de las pruebas.

3.3.1. Visión General. AGEDIS (**A**utomated **G**eneration and **E**xecution of Test Suites for **DI**stributed Component-based **S**oftware) es una amplia metodología que abarca tanto la generación de pruebas como la ejecución automática de las mismas, para lo cual ha sido desarrollada una serie de herramientas para la ejecución y automatización de estos puntos.
AGEDIS propone generar una serie de diagramas que presentarán la arquitectura del sistema de una manera más amplia y específica para lo que utiliza una serie de anotaciones propias y algunas especificaciones extras, de manera que se pueda generar no solo la aplicación sino además una serie de pruebas necesarias para la optimización del resultado de la misma. En vista de esto se pretende generar una serie de herramientas vinculadas a una misma interfaz que permita desarrollar fácilmente todos los pasos necesarios para este fin.

3.3.2. Procedimiento para la obtención de pruebas. AGEDIS propone generar las pruebas a partir de un modelo del sistema, para lo cual precisa seguir estos pasos:

3.3.2.1. Elaboración del modelo del sistema. El modelo del sistema que AGEDIS propone elaborar debe ser basado en los diagramas UML de clases para estructurar el sistema, de estados (se propone uno de estados por cada clase) para modelar el comportamiento de los objetos instanciados a partir de dicha clase, y, de ser necesario, uno de objetos que modelaría el estado inicial del sistema. Estos diagramas seguirían la notación UML aunque con algunas características propias de AGEDIS, como, por ejemplo, la implementación de estereotipos tales como initial, start, finish, include y exclude, en los diagramas de objetos [7].

3.3.2.2. Creación de directivas de pruebas. Las directivas de pruebas deben definir los objetivos de las pruebas a generar, tales como carga del sistema, el funcionamiento del mismo o la manera en que se manejará la información tanto en su flujo como en las

características de la misma (longitud, tipo, entre otras). Para generar estas directivas se propone crear diagramas de objetos y de estados que definan tanto las características de las clases en cada momento de las pruebas como las interacciones entre ellas.

3.3.2.3. Generación del catálogo de pruebas. Este es un paso que las herramientas de AGEDIS generan automáticamente a partir de la información de los dos pasos anteriores.

3.3.2.4. Revisión del catálogo de pruebas. Opcionalmente se podrá revisar el catálogo de pruebas generado automáticamente, de manera que si alguna prueba no concuerda con los requerimientos se deberá volver al paso uno o dos, según sea el caso para corregir las fallas que se hayan presentado.

3.3.2.5. Ejecución del catálogo de pruebas. Al igual que el paso 3, este proceso es ejecutado automáticamente por las herramientas, lo cual permite ejecutarlo cuantas veces sea necesario por modificaciones en la aplicación o por nuevas directivas de ejecución de las pruebas. Este proceso genera un informe de resultado de las pruebas que podrá ser analizado a través de las herramientas de AGEDIS.

3.3.2.6. Análisis de las pruebas. En este paso se podrá verificar el resultado de las pruebas revisando que sea acorde a lo esperado y que las pruebas hayan sido ejecutadas correctamente, de manera que si algo llegase a faltar o fallar se podrán modificar las directivas de las pruebas, o, en caso extremo, el modelo del sistema.
Para soportar la automatización de estos pasos, AGEDIS ha generado una serie de herramientas que se describen brevemente a continuación.

3.3.3. Arquitectura de AGEDIS. Estos pasos son soportados por una serie de herramientas cuya arquitectura es [9]

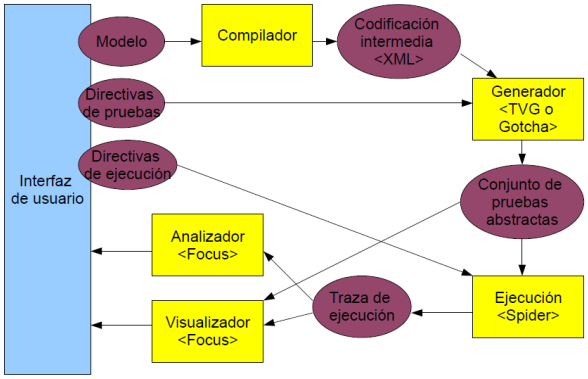

Figura 7. Arquitectura de AGEDIS

3.3.3. Herramientas propias de AGEDIS. Las herramientas creadas por AGEDIS son [8]:

- **Objecteering:** Esta herramienta se utiliza para construir los modelos de sistema y las directivas de ejecución de pruebas, utilizada específicamente en los pasos uno y dos.
- **Gotcha y TVG:** Son dos herramientas que funcionan para generar las pruebas a partir de la codificación del modelo obtenido a partir de objecteering, utilizadas en el paso tres.
- **Spider:** Es la herramienta encargada de la ejecución automática de las pruebas a partir de las directivas de ejecución y del conjunto de pruebas generadas por Gotcha y TVG, utilizada en el paso cinco.
- **Focus:** Esta herramienta permite visualizar y analizar las pruebas generadas por Gotcha o TVG, y, además, el resultado de las pruebas generado por Spider. Es utilizada en los pasos cuatro y seis.

4. Conclusiones

En este capítulo hemos presentado alternativas metodológicas y técnicas para la automatización de la generación de casos de prueba a partir de modelos. Específicamente hemos revisado tres opciones todas basadas en el modelo de estado (notación UML) del sistema. SCENT y UML-Based Statistical parten de los casos de uso y de la descripción ampliada de los mismos y AGEDIS de los diagramas de clases. SCENT introduce un nuevo estereotipo, los diagramas de dependencia y UML_Based Statistical otro estereotipo, los diagramas de uso (con probabilidades).

5. Trabajo Futuro

Debido al mayor cubrimiento que tiene AGEDIS en la automatización, ya que cubre el ciclo completo: modelo del sistema, generación de los casos de prueba, ejecución del catálogo de pruebas, análisis de los resultados de las pruebas con los resultados esperados y que además incluye una serie de herramientas para apoyar el proceso, queremos investigar tanto la conceptualización teórica o de modelamiento en cada uno de los pasos como de ser posible, las herramientas de apoyo.

6. Agradecimiento

Agradecemos al ingeniero Nelson Suárez, coordinador de control calidad y pruebas de sistemas corporativos del Grupo AVAL – ATH, por su interés

en el tema, la documentación proporcionada y referida y por su orientación en los temas de investigación. Esperamos que esta reseña sirva de base para futuras investigaciones de aplicabilidad en esta y otras organizaciones.

7. Referencias

[1] J. Ryser, M. Glinz, "A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts", Institut für Informatik, University of Zurich 1999.

[2] J. Ryser, M. Glinz, "SCENT - A Method Employing Scenarios to Systematically Derive Test Cases for System Test", Institut für Informatik, University of Zurich 2000.

[3] J. Ryser, M. Glinz, *A Practical Approach to Validating and Testing Software Systems Using Scenarios*, Institut für Informatik, University of Zurich 1999.

[4] H, Hong, I, Lee, O, Sokolsky, *Automatic Test Generation from Statecharts Using Model Checking*, University of Pennsylvania, 2001.

[5] M, Riebisch, I, Philippow, M, Götze, *UML-Based Statistical Test Case Generation*, Technical University, Ilmenau, Germany, 2002.

[6] G., Walton, J, Poore, C. Trammel, *Statistical testing of software based on a usage mode,* University of Tennessee, Knoxville, 1995

[7] C, Crichton, A, Cavarra, J, Davies, *Using UML for Automatic Test Generation*, Oxford University Computing Laboratory, 2001

[8] A. Hartman, K. Nagin, *The AGEDIS Tools for Model Based Testing*, Haifa University, Israel, 2004

[9] A. Hartman, *AGEDIS architecture*, Haifa University, Israel, 2001