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

Henry Roberto Umaña Acosta – Profesor Asociado
Miguel Cubides González – Investigador

Pruebas del software

- Constituyen una *verificación dinámica* del comportamiento de un programa
- No sólo abarcan el diseño y el código sino además las librerías, el S.O., redes, etc..
- La pruebas exhaustivas no son posibles dado el número infinito de posibles secuencias de entradas válidas e inválidas

Alcance de las pruebas

- Se debe escoger un número finito de casos de prueba que expongan las fallas del sistema
- Existen diferentes estrategias para determinar los casos de prueba y los datos de prueba como por ejemplo *Particiones de Equivalencia*
- Debe incluir la especificación del comportamiento esperado (Oráculo)

Tipos de pruebas

- *Por alcance*: unitarias, de componentes, de integración
- *Por características*: funcionales, desempeño, usabilidad, etc..
- *Por clase de información disponible*: de caja negra, de caja blanca
- Las *pruebas basadas en modelos* están dirigidas a la automatización de pruebas funcionales de componentes y de integración de caja negra

Modelos

- El modelo del SUT (System Under Test) debe ser pequeño en relación con el sistema, pero debe cubrir adecuadamente sus características fundamentales
- Las herramientas de pruebas basadas en modelos son utilizadas para generar automáticamente las pruebas a partir de ese modelo.

Herramientas

- Las salidas de esas herramientas constituyen las *pruebas abstractas*: la secuencia de operaciones de pruebas con los datos de prueba asociados y los valores esperados de salida (oráculo)
- El siguiente paso es la *concretización*: la transformación de estas salidas en scripts ejecutables (JUnit, Python, por ej.)
- La ejecución de estos scripts puede ser controlada y monitoreada por una herramienta de ejecución de pruebas

El proceso de pruebas basadas en modelos (MBT: Model Based Testing)

- Modelar el SUT (System Under Test)
- Generar las pruebas abstractas desde el modelo
- Concretizar las pruebas abstractas para hacerlas ejecutables
- Ejecutar las pruebas y registrar los resultados
- Analizar los resultados de las pruebas

Beneficios de MBT

- MBT no es gratis: hay que elaborar el modelo.
- El beneficio es que se genera automáticamente el conjunto de pruebas en una forma más exhaustiva y económica que en la generación y ejecución manual de los mismos.
- Adicionalmente se cuenta con una base para involucrar el cambio en el sistema a través de los cambios en el modelo

Testimonio

- “I recently worked in a scrum project, where the team made the decision to automate all tests. In fact, the Definition Of Done was: only demo when functionality is fully developed, unit tests exists (with 85% code coverage), and every test that verifies the requirements is fully automated, and can run in a nightly build.

  This we have achieved. But, we (the testers) could never have done this without using MBT as a technique creating and maintaining all the automated tests. All the logic, all the design effort was invested and placed in the models. This made us very agile. Changes were made in the models, and, if necessary, in the test execution tools.”

  Kristian Karl <kristian.hermann.karl@gmail.com>
- jueves, 3 diciembre, 2009

Experiencias en IBM

- *GOTCHA-TCBeans*: Genera las pruebas abstractas a partir de un modelo de máquina de estado finito. *TCtranslator* concretiza las pruebas abstractas en scripts en Java y en C.
- Usado en la prueba de un subsistema de POSIX. Manualmente el esfuerzo fue de 12 personas-mes y se encontraron 18 defectos.
- Con las herramientas el esfuerzo fue de 10 personas-mes y se encontraron 2 defectos adicionales y de los 18 defectos iniciales, 15 se encontraron más temprano

Experiencias en Microsoft

- *Test Model Toolkit:* Desarrollada por el equipo de Internet Explorer y basadas en modelos de máquinas de estado finito
- Usada por el equipo de BizTalk que por ejemplo gastó 8 semanas-hombre para escribir un conjunto de pruebas manualmente, pero con las herramientas gastó una semana-hombre, aumentando el cubrimiento en un 50%.

Limitaciones de las pruebas basadas en modelos (MBT)

- No se garantiza la sincronización total entre el modelo del SUT y la implementación
- Requiere habilidades y conocimientos especiales para construir los modelos, aumentando los costos iniciales y la curva de aprendizaje
- Sólo se usa para pruebas funcionales. No cubre los requerimientos no funcionales

Modelando el sistema

- El modelo de pruebas no es estrictamente igual al modelo del sistema
  - Obviar información innecesaria para las pruebas
    - Clases
    - Operaciones
    - Relaciones
  - Agregar relaciones en las operaciones del modelo de pruebas

Principales notaciones para modelado

- Notación Pre/Post (B, OCL, Z)
- Notación basada en transición (FSM, UML State Machines)
- Notación basada en historia (UML Sequence Diagrams)

Metodologías estudiadas

- SCENT
- UML-Based Statistical Test Case Generation
- AGEDIS

SCENT (Method for **SCEN**ario-Based Validation and **T**est of Software)

- Visión General
  - Objetivo
    - Integrar las pruebas desde la fase de análisis
    - Aprovechar los resultados obtenidos en la fase de análisis para futuras fases
  - Desarrollo
    - Basado en escenarios
    - Generación de diagramas de estados
  - Características especiales
    - Formalización de escenarios
    - Diagramas de dependencia

Procedimiento

- Creación de escenarios
  - Desarrollo de diagramas de dependencia
- Formalización de escenarios
  - Generación de diagramas de estados a partir de los escenarios
    - Información extra:
      - Precondiciones
      - Entradas/salidas y rango de datos
      - Requerimientos no funcionales
- Derivación de los casos de prueba
  - A partir de diagramas de estados
  - A partir de diagramas de dependencias
  - Pruebas adicionales

Ejemplo (ATM, SCENT)

- Especificación:

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

Diagrama de estados

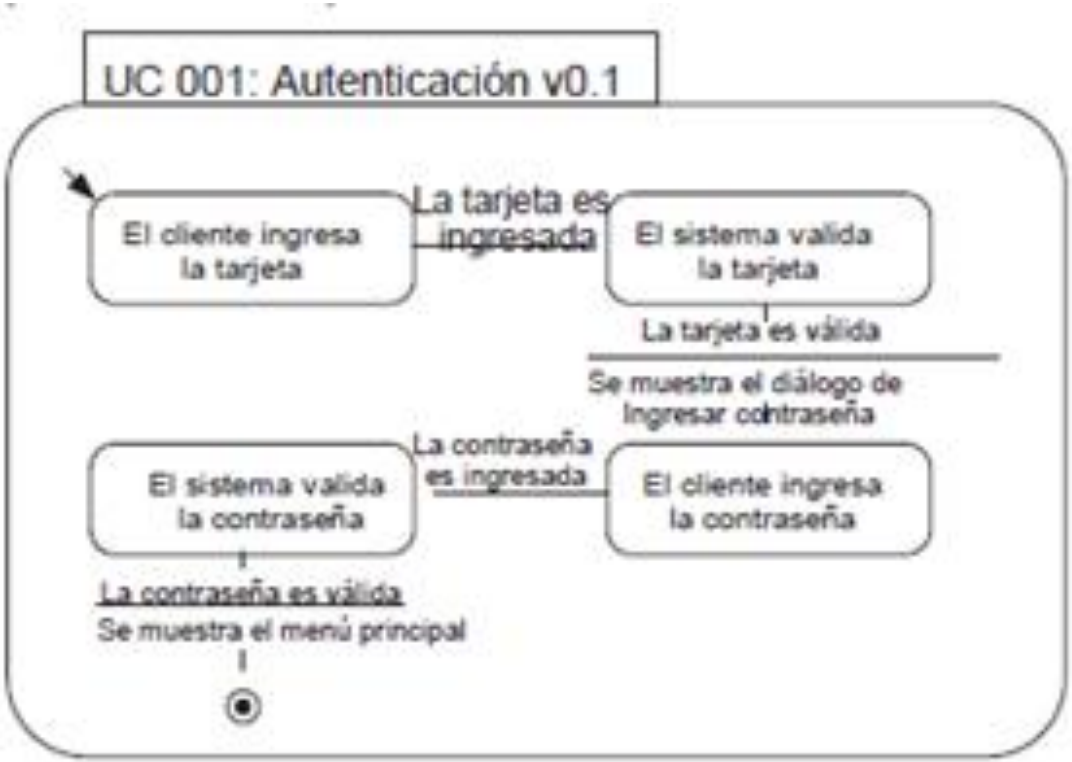


Diagrama de dependencias

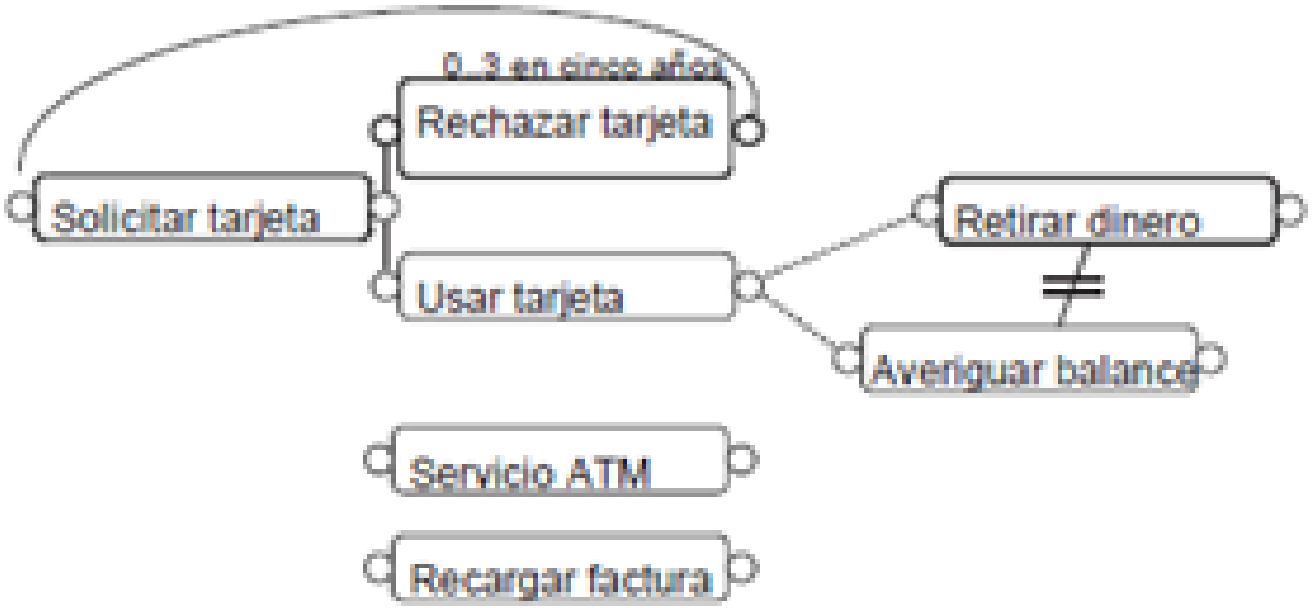
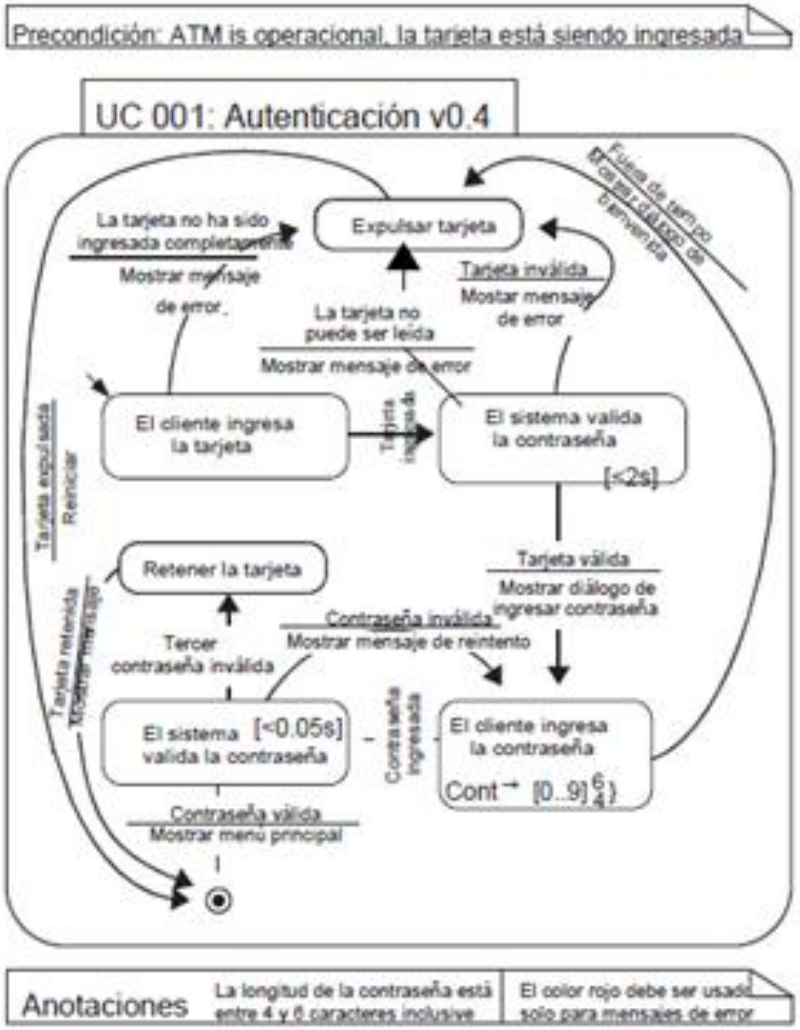


Diagrama de estados extendido



Pruebas generadas

ID	ESTADO	ENTRADAS DE USUARIO	SALIDA ESPERADA
1.1	Tarjeta deslizada	La tarjeta puede ser leída, la tarjeta es válida, el PIN es válido (1234) y ha sido ingresado en dentro del tiempo límite	Menú principal mostrado
1.2	Tarjeta deslizada	La tarjeta puede ser leída, la tarjeta es válida, el PIN es inválido (124) y ha sido ingresado dentro del tiempo límite	Mensaje de reintento mostrado
1.3	Mensaje de reintento	PIN inválido (123) ingresado dentro del tiempo límite, segundo intento	Mensaje de reintento mostrado
1.4	Mensaje de reintento	PIN inválido (123) ingresado dentro del tiempo límite, tercer intento	La tarjeta es retenida, informándole al usuario
…	…	…	…

UML-Based Statistical Test Case Generation

- Visión General
  - Objetivo
    - Integrar las pruebas desde la fase de análisis
    - Generar las pruebas basados en una ponderación de probabilidad sobre la ocurrencia de cada evento en el modelo del sistema
  - Desarrollo
    - Basado en diagramas de estados desarrollados a partir de los casos de uso
  - Características especiales
    - Diagramas de uso
    - Modelos de uso

Procedimiento

- Refinado de casos de uso
- Desarrollo de diagramas de estado a partir de los casos de uso
  - Uno por cada caso de uso
  - Uno general
- Desarrollo de diagramas de uso a partir de diagramas de estados
- Desarrollo de modelos de uso a partir de diagramas de uso
- Generación de casos de prueba a partir de los modelos de uso

Ejemplo

Nombre	Prestar libro
Propósito	Presta un libro de esta librería a un usuario de esta librería por un tiempo limitado
Actores	Bibliotecario
Pre-condiciones	El sistema está corriendo. El usuario es válido en esta librería.
Post-condiciones	El libro es prestado al usuario. Una fecha de retorno es definida
Flujo principal	1. El bibliotecario abre el diálogo de selección de usuario, el usuario seleccionado es mostrado. 2. El bibliotecario abre el diálogo de préstamo de libros, la fecha de retorno es exhibida. 3. El bibliotecario confirma la acción 4. El sistema marca el libro como prestado y lo agrega a la lista de libros prestados del usuario.
Extensiones	2-a. El usuario debe pagar una multa 2-a.1. El sistema muestra un diálogo informando que no se puede prestar ningún libro hasta que se pague la deuda y cambia al diálogo de caja. 2-a.2. Después de pagar exitosamente, el sistema retorna al diálogo de préstamo. 2-a.3. Si no se paga exitosamente, el sistema muestra el diálogo de selección de usuario. 2-b. El libro solicitado está prestado o es de referencia únicamente. 2-b.1. El sistema muestra un diálogo con la información y retorna al diálogo de préstamo de libros. 3-a. El bibliotecario cancela la acción 3-a.1. El sistema retorna al diálogo de préstamo de libros.
Casos de uso incluídos	Muestra de libros prestados por un usuario; Recolección de pagos para un usuario.

Diagrama de estado

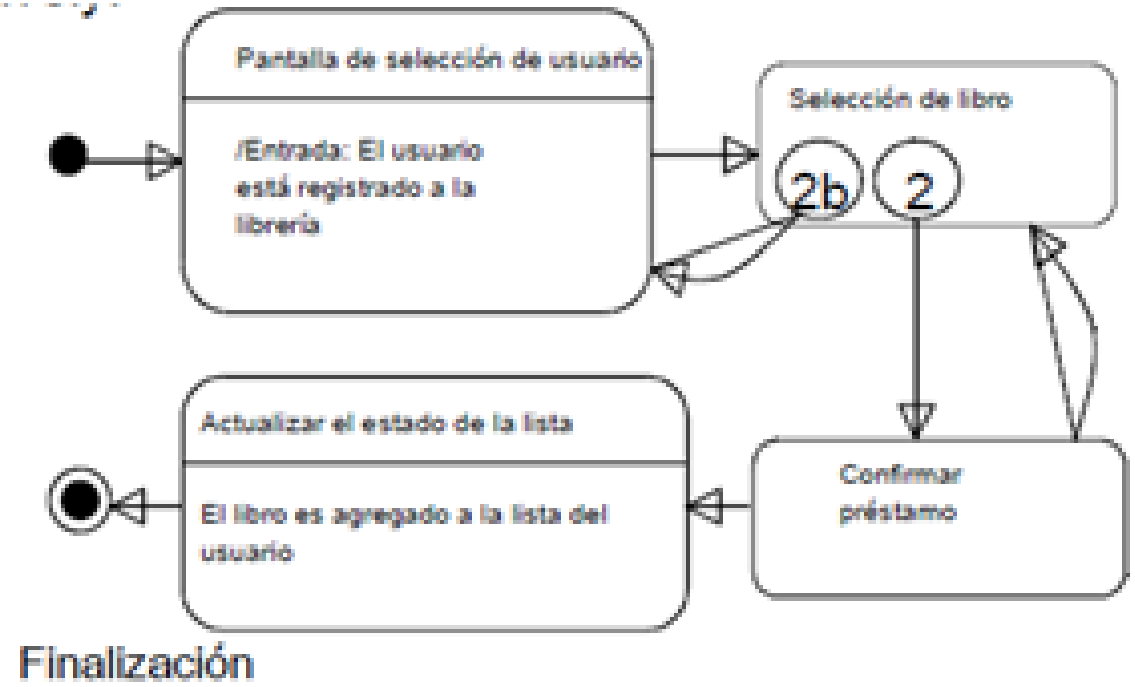
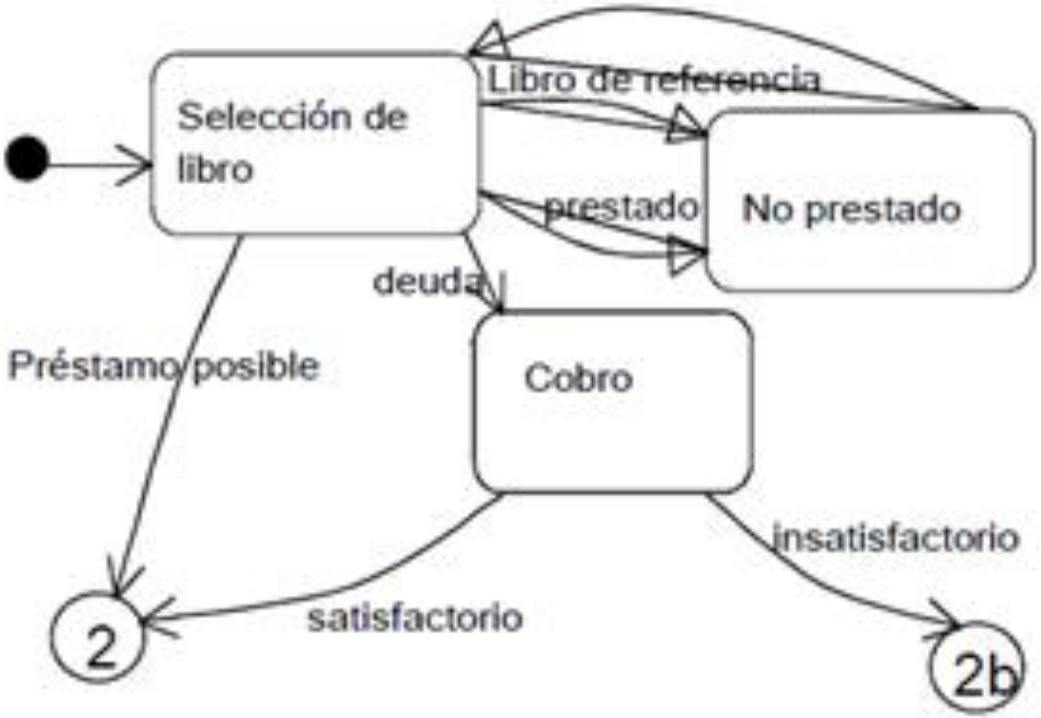
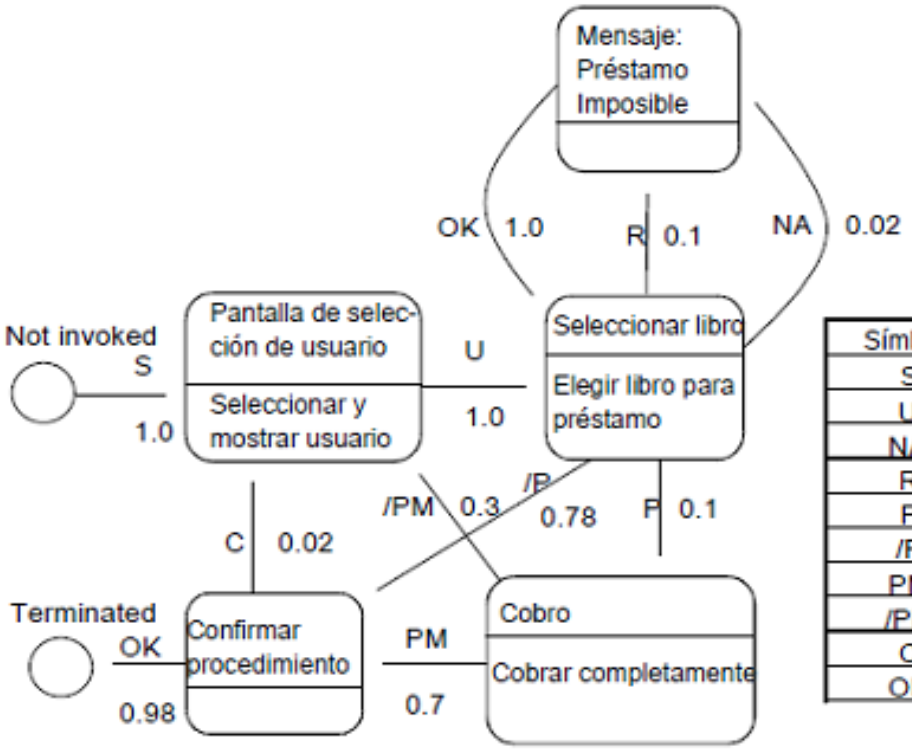


Diagrama de uso



Modelo de uso



Símbolo	Descripción
S	Seleccionar los libros a prestar
U	Usuario seleccionado
NA	Libro no disponible
R	Libro exclusivo de referencia
P	El usuario tiene deuda a pagar
/P	Sin deudas, préstamo posible
PM	Pago satisfactorio
/PM	Pago insatisfactorio
C	Cancelado por bibliotecario
OK	Confirmado por bibliotecario

Caso de prueba base:
S U R OK NA OK P /PM U P PM C U /P OK
Caso de prueba usuario con deuda, pagada en el segundo intento:
S U P /PM U P PM OK

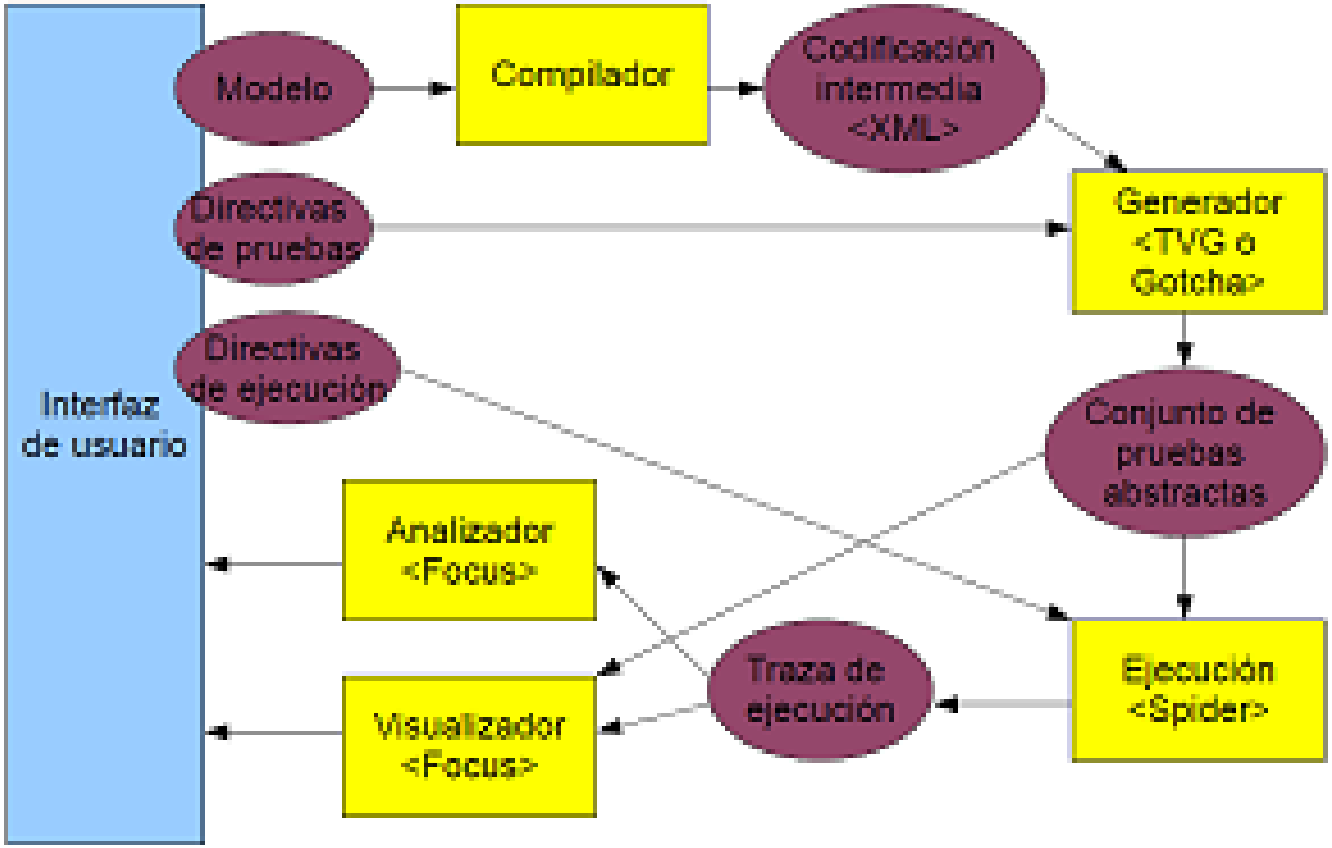
AGEDIS

- Visión General
  - Objetivo
    - Integrar las pruebas desde la fase de análisis
    - Crear herramientas para desarrollar y automatizar el desarrollo y ejecución de las pruebas
  - Desarrollo
    - Basado en diagramas UML
      - Clases
      - Estados
      - Objetos
  - Características especiales
    - Desarrollo e implementación de AML
    - Implementación de IF
    - Desarrollo e implementación de herramientas propias para la automatización de las tareas

Procedimiento

- Elaboración del modelo del sistema
  - AML
- Creación de las directivas de pruebas
- Generación del catálogo de pruebas
- Revisión del catálogo de pruebas
- Ejecución del catálogo de pruebas
- Análisis de las pruebas

Arquitectura de AGEDIS



Conclusiones

Metodología	Parten de:	Características especiales
SCENT	Casos de uso	Formalización de escenarios Diagramas de dependencia
UML-Based-Statistical	Casos de uso	Diagramas de uso Modelos de uso
AGEDIS	Diagrama de clases	AML Herramientas propias

Agradecimientos

- 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

Bibliografía

- J. Ryser, M. Glinz, “A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts”, Institut für Informatik, University of Zurich 1999
- J. Ryser, M. Glinz, “SCENT - A Method Employing Scenarios to Systematically Derive Test Cases for System Test”, Institut für Informatik, University of Zurich 2000
- J. Ryser, M. Glinz, *A Practical Approach to Validating and Testing Software Systems Using Scenarios*, Institut für Informatik, University of Zurich 1999
- H, Hong, I, Lee, O, Sokolsky, *Automatic Test Generation from Statecharts Using Model Checking*, University of Pennsylvania, 2001
- M, Riebisch, I, Philippow, M, Götze, *UML-Based Statistical Test Case Generation*, Technical University, Ilmenau, Germany, 2002
- G., Walton, J, Poore, C. Trammel, *Statistical testing of software based on a usage mode,* University of Tennessee, Knoxville, 1995
- C, Crichton, A, Cavarra, J, Davies, *Using UML for Automatic Test Generation,* Oxford University Computing Laboratory, 2001
- A. Hartman, K. Nagin, *The AGEDIS Tools for Model Based Testing,* Haifa University, Israel, 2004
- A. Hartman, *AGEDIS architecture,* Haifa University, Israel, 2001