Proyecto GxUnit

Construcción de una herramienta para automatización de pruebas unitarias en GeneXus

Enrique Almeida	**Alejandro Araújo**	**Uruguay Larre Borges**
Concepto	FYMNSA	GeneXus Consulting
Montevideo,Uruguay,11800	Montevideo,Uruguay,11200	Montevideo,Uruguay,11200
ealmeida@concepto.com	alar758@gmail.com	ularre@genexusconsulting.com

Abstract

The goal of GxUnit project is the construction of an automatization tool for unit proofs associated to GeneXus. Such project is exposed in this paper, also the products obtained from independent and paralel development processes that correspond to the initial versions of the tool, made by two student groups attendees of the "Software Engineering Project 2007" dictated by *Facultad de Ingeniería de la Universidad de la República del Uruguay*, where the authors played de Customer role. Both versions are distinguished by the denominations of GxUnit1 and GxUnit2.

Keywords: FIT, GeneXus, Software Engineering, Unit Testing, xUnit.

Resumen

El objetivo del proyecto GxUnit es la construcción de una herramienta para la automatización de pruebas unitarias asociada a GeneXus. En este informe se expone acerca de dicho proyecto y de los productos resultantes de dos procesos de desarrollo independientes y paralelos que dieron lugar a las versiones iniciales de la herramienta, llevados adelante por dos grupos de estudiantes del curso "Proyecto de Ingeniería de *Software* 2007", de la Facultad de Ingeniería de la Universidad de la República del Uruguay, para los cuales los autores de este informe actuamos en el rol de Clientes. Dichas versiones se distinguen con las denominaciones GxUnit1 y GxUnit2.

Palabras claves: FIT, GeneXus, Ingeniería de Software, Prueba Unitaria, xUnit.

1. GENEXUS

El marco para la automatización de pruebas unitarias propuesto se asocia a GeneXus[1] [7], herramienta de especificación de sistemas de información basada en la aplicación de un modelo matemático que permite capturar e integrar las visiones de los usuarios en bases de conocimiento (KB[2]) a partir de las cuales genera, mediante ingeniería inversa y procesos de inferencia, bases de datos normalizadas y aplicaciones completas; para diferentes plataformas de destino a partir de una misma especificación básica, pudiendo mantenerlas en forma automatizada ante cambios en los requerimientos [5] [9] [21] [23]. GeneXus modela la realidad mediante un conjunto de instancias de "objetos tipos"[3] almacenados en las KBs. Produce el código necesario para implementar las aplicaciones en diferentes lenguajes y plataformas de destino mediante la utilización de programas generadores. Se apoya en una metodología incremental e iterativa y trabaja sobre especificaciones, lo cual permite independencia tecnológica. Su última versión (versión X) posee una arquitectura extensible, por lo cual pueden agregarse al producto paquetes de *software*, conocidos como extensiones, capaces de interactuar con la base de conocimiento, pudiendo ser desarrolladas por terceros [8].

2. EL PROYECTO GXUNIT

2.1 Fundamentación

Una parte considerable del total de *software* producido en Uruguay se elabora con esta herramienta creada y mantenida por una empresa uruguaya, Artech. En el área de pruebas dicha herramienta no brinda actualmente una funcionalidad específica que se aproxime en versatilidad y potencia al resto de sus características, por lo cual se entendió oportuno incorporarle mecanismos para la automatización de las pruebas unitarias, de valor para el Analista GeneXus[4].

2.2 Antecedentes

La propuesta GxUnit tuvo su origen en el año 2003 [1] cuándo se anuncia el objetivo de crear una herramienta de automatización de pruebas unitarias en GeneXus. En el año 2004 se formalizó la propuesta [2] caracterizándose la misma en la línea de las herramientas xUnit[5]. Los objetivos básicos planteados fueron los siguientes: crear un marco de pruebas asociado a GeneXus, poder escribir las pruebas en GeneXus, ejecutar las pruebas y registrar los resultados.

Considerando para el desarrollo un enfoque iterativo e incremental se postuló ir cumpliendo con el objetivo de generar objetos GeneXus para pruebas de instancias de los siguientes objetos GeneXus: Objetos sin UI[6]: *Procedures, Business Transactions*; objetos con UI: *WebPanels* (pantallas de consulta *web*), *Transacciones (Transactions), WorkPanels* (pantallas de consulta *win*).

En el año 2006 se retoma bajo la forma de "Proyecto Colaborativo GeneXus" [3] [4] para finalmente en julio de 2007 proponer el proyecto para su desarrollo en el ámbito del curso "Proyecto de Ingeniería de *Software* 2007" de la Facultad de Ingeniería de la Universidad de la

[1] Producida en Uruguay por la empresa Artech, su primera versión fue liberada al mercado para su comercialización en 1989. Sus creadores, Breogán Gonda y Nicolás Jodal, han sido galardonados con el Premio Nacional de Ingeniería en 1995 otorgado por el "Proyecto GeneXus" [10]
[2] *Knowledge Base*
[3] No se refiere a objetos en el sentido de OOLP (lenguajes de programación orientada a objetos). Se refiere a objetos GeneXus tales como *Transactions, WebPanels, WorkPanels, Procedures*, etc.
[4] De ahora en adelante su utilizará el término "Analista" como sinónimo de "Analista GeneXus"
[5] Familia de *Frameworks* para pruebas unitarias (JUnit [14] y otros)
[6] Interfase de usuario

República del Uruguay. Entre agosto y noviembre de dicho año dos grupos de estudiantes desarrollaron en forma independiente y en paralelo, dos versiones iniciales de la herramienta GxUnit en el contexto del mencionado curso [15]. Se actuó en el rol de Cliente, realizándose el seguimiento de la construcción y validación de los prototipos. Los productos de *software* resultantes de dichos desarrollos han sido liberados al dominio público y pueden ser descargados desde Internet [11] [12].

3 LA HERRAMIENTA PARA PRUEBAS UNITARIAS GXUNIT

3.1 Características generales

La herramienta debe permitir definir y especificar instancias de objetos para prueba unitaria ("*fixtures*") de otras instancias de "objetos tipos" GeneXus del SUT[7]. Los objetos para prueba recogen la especificación de la prueba a implementar y a partir de los mismos se genera el código necesario para ejecutarla en diferentes plataformas. En la figura 1 se ejemplifica al respecto de lo anteriormente expuesto, observándose la relación entre los objetos de prueba y los objetos del SUT y los correspondientes programas generados en los diferentes lenguajes para las distintas plataformas de destino. La especificación para las pruebas es recogida desde parámetros contenidos en tablas. El resultado de las pruebas se almacena.

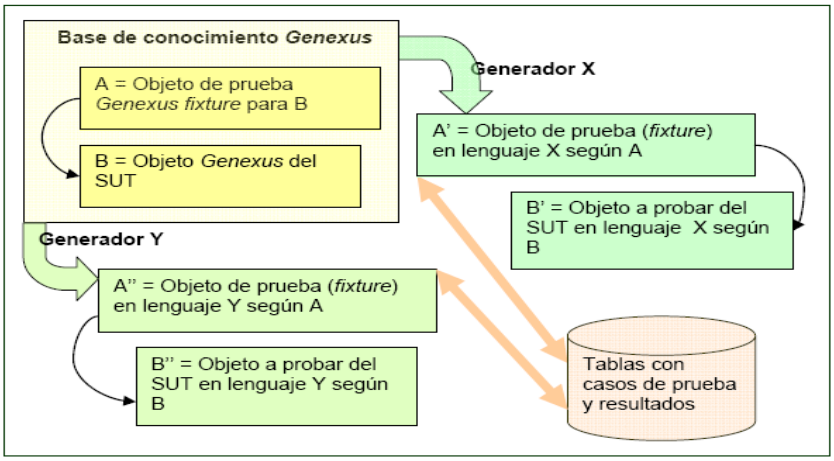

Figura 1: Objetos para prueba unitaria.

Las instancias de objetos para prueba se asocian a las instancias de objetos[8] correspondientes a probar y son sensibles a los impactos de los cambios en los mismos, automatizándose su reconstrucción.

3.2 Especificación

La especificación de GxUnit ha recibido aportes de diversas fuentes. En particular se recogió la propuesta de implementar un formato de especificación de las pruebas apoyado en ideas contenidas en el "*Framework for Integrated Test*" (FIT) [6] [20]. La elección de un formato tabular para la implementación de las pruebas permite la especificación sustentable de requerimientos, por la vía de ejemplos, de forma que complementa la comunicación y fomenta la colaboración entre

[7] *System Under Test* (sistema sometido a pruebas)
[8] De ahora en adelante se utilizará el termino "objetos" para referirse tanto a los "objetos tipos GeneXus" como a sus instancias particulares dentro del modelo de una aplicación en una KB

desarrolladores, verificadores y usuarios. El valor de este tipo de representaciones ha sido reconocido y utilizado desde hace largo tiempo [13]. Simultáneamente, se aprovechan las características de GeneXus para facilitar y resolver el costoso mantenimiento de los casos de prueba que se presentan cuándo se utilizan herramientas del tipo de FIT y la intensa reelaboración ante los cambios en los casos de prueba y el SUT, la sobrecarga en los equipos de desarrollo y otros problemas de escalabilidad [16] [19] [17] [18] [22] [24].

Para el proyecto inicial se suministraron a ambos grupos la siguiente visión, misión y los siguientes requerimientos, expuestos para este informe bajo la forma de relatos (*stories*). El alcance se negoció y acotó con cada uno de los grupos en forma independiente.

Visión: El propósito es brindar al Analista un mecanismo para automatizar la creación, mantenimiento y ejecución de pruebas unitarias.
Misión: Construir una herramienta que permita automatizar las pruebas unitarias de objetos GeneXus a ser utilizada por Analistas.

3.2.1 Consideraciones técnicas

La herramienta se deberá construir como una extensión para GeneXus. Deberá tener un diseño modular. Es preciso tener muy en cuenta que esta es una etapa inicial de la herramienta, que irá desarrollándose en incrementos. A futuro se espera poder incorporarle componentes que permitan generar pruebas para otros objetos GeneXus que no serán contemplados en esta primera edición y nuevos editores para los casos de prueba. El reuso, la independencia entre capas y un bajo acoplamiento son requisitos importantes. Se deberá programar en C# y se utilizará el SDK[9] brindado por Artech para la programación de extensiones.

3.2.2 Relatos

- El Analista definirá "casos de prueba" desde el IDE[10] de GeneXus, desde donde también podrá comandar su ejecución y ver los resultados. Se deberá suministrarle un editor para efectuar el ingreso de estos casos de prueba.
- El Analista podrá ejecutar un conjunto de pruebas en modo desatendido, por ejemplo luego de una construcción (*build*), adicionalmente a las ejecuciones que realiza desde la interfase.
- Los casos de prueba se almacenarán en la KB del SUT.
- El resultado de la ejecución de las pruebas no se almacenará en la KB.
- Se desea que la herramienta tenga una interfase similar a los productos xUnit para la ejecución y visualización de primer nivel de los resultados. Se desea también que esté acorde a la interfase del IDE de GeneXus, para que no resulte extraña al Analista.
- Se definen como objetos verificables aquellos pertenecientes al dominio de objetos para los cuales GxUnit puede generar y ejecutar pruebas unitarias. En esta primera instancia estos serán solamente objetos de tipo *procedure*.
- La herramienta generará en forma automatizada objetos verificadores de tipo *procedure* (*tests*) a partir de los objetos verificables y de los casos de prueba. Serán los encargados de guiar la prueba invocando a los objetos a verificar.
- El Analista necesita poder conocer rápidamente cuales son los objetos verificables y cuales de estos tienen casos de prueba ya definidos.

[9] Software Development Kit
[10] Integrated Development Environment

- Se deberá implementar un oráculo que permitirá saber si cada prueba fue exitosa o fallida. El Analista podrá indicar que se realice una comprobación parcial, lo cual implica que habrá resultados de la prueba en este caso que no influirán en la decisión acerca de si la misma pasa o falla.
- Las pruebas fallidas se marcarán en Rojo, las que pasaron en Verde, aquellas no ejecutadas en Gris. De capturarse excepciones estas se desplegarán pintadas de Amarillo.
- Existirán *procedures* (procedimientos) "verificadores del usuario" (PVU), que escribirá el Analista, y que podrán ser asociados a las pruebas para ser invocados previo a su finalización. Estos procedimientos implementarán oráculos y poseerán un único parámetro de tipo "*booleano*" para indicar si la prueba fue exitosa o fallida.
- El Analista definirá una prueba indicando: el objeto verificable, uno o varios PVUs y un conjunto de datos ("tuplas o casos de prueba") de entrada y resultados esperados que deberán corresponderse respectivamente con cada uno de los parámetros de entrada y de salida del objeto verificable.
- El editor del caso de prueba deberá presentar una grilla "inferida" a partir de los parámetros de entrada y salida del objeto verificable para que el Analista ingrese o modifique los casos de prueba (datos de entrada y resultados esperados).
- Para un mismo objeto verificable se pueden definir varios "casos de prueba". El Analista podrá indicar cuales de estos casos desea incluir en la próxima prueba y cuales desea ignorar.
- Durante la ejecución de una prueba se iterará la invocación del objeto a verificar, una vez por cada "caso de prueba" participante en la prueba, pasándole como parámetros de entrada los datos de entrada indicados en un "caso de prueba". Al finalizar la prueba se invoca a los PVUs (de existir).
- Dada una prueba, el Analista deberá poder visualizar, para cada "caso de prueba" los datos de entrada, los resultados esperados y datos de salida obtenidos. También necesitará acceder al resultado de ejecuciones anteriores.
- Una prueba se considera fallida si produce resultados no coincidentes con los esperados y/o si el PVU indica falla. La comparación de los resultados esperados con los obtenidos, para el alcance de este desarrollo, se limitará a la igualdad.
- Debe ser posible, para un Analista, ofrecer la posibilidad de cargar los datos de entrada y resultados esperados, desde un archivo externo.
- Los tipos de datos a considerar son los básicos de GeneXus, incluyendo tipos de datos estructurados (SDTs[11]).
- El resultado de la ejecución de las pruebas se podrá almacenar con diferentes niveles de detalle.
- Los cambios que se produzcan a nivel de los parámetros de un objeto verificable deberán impactarse contra su(s) caso(s) de prueba, detectándose automáticamente las diferencias para permitir tanto su reconstrucción automatizada como la de los *tests*.
- La eliminación de PVUs implicará advertir al Analista acerca de los casos de prueba que los utilicen y reconstruir en forma automatizada los casos de prueba y los *tests*.
- Debe existir un mecanismo para eliminar fácilmente un objeto verificable con casos de prueba asociados. Dicha eliminación también eliminará sus casos de prueba y sus *tests*.
- Al Analista le interesará verificar el estado de la base de datos (BD) luego de la ejecución de una prueba, por lo cual se desea proveer un mecanismo en tal sentido que le permita conocer la variación Δ BD = BDinicial - BDfinal y verificar que el estado final sea el esperado.

[11] *Structured Data Types*

3.3 Implementaciones.

Se obtuvieron como resultantes principales del proyecto dos productos de *software*, que se denominaron GxUnit1 y GxUnit2. El diseño de la solución y la documentación de la misma estuvieron a cargo de cada uno de los grupos de estudiantes.

3.3.1 Principales características

A partir del mismo conjunto de requerimientos cada grupo desarrolló una herramienta con diferencias importantes en cuanto al diseño de la solución. Las semejanzas están dadas por haber satisfecho un núcleo común de requerimientos y también por algunas soluciones de diseño común, como ser el almacenamiento de los resultados de las pruebas en formato XML[12] y la utilización de *Web Services* (WS) para comandar su ejecución desde la extensión. Las diferencias surgen en el diseño de las interfases, su vinculación con los objetos GeneXus, la generación de procedimientos GeneXus para prueba y el almacenamiento de los resultados. También surgen diferencias debidas al alcance acordado con cada grupo de estudiantes. Se enumeran a continuación las principales características de cada implementación.

3.3.2 GxUnit1

- Genera un objeto verificador por cada objeto verificable Estos procedimientos verificadores pueblan la KB y se corresponden uno a uno con la prueba y el objeto a verificar. No presentan restricciones en la cantidad de parámetros a utilizar.
- El objeto verificador se implementa como un WS que es consumido desde la extensión
- Crea una "parte"[13] nueva para todo procedimiento En dicha parte se muestra el editor del caso de prueba, en el cual pueden incluirse las "tuplas o casos de prueba" de valores de entrada y de salida.
- Almacena los datos para prueba y los resultados en archivos XML.
- Ofrece una primera aproximación a la verificación de la BD.
- Admite procedimientos con parámetros de tipo datos estructurados (SDT).
- Permite reconstruir un caso de prueba ante cambios en la regla *parm* del procedimiento a verificar minimizando la pérdida de datos.

3.3.3 GxUnit2

- Genera un único objeto verificador, capaz de ejecutar todas las pruebas. Dicho objeto utiliza invocaciones dinámicas de GeneXus para ejecutar los procedimientos a probar. Necesitaría para lograr su implementación completa poder pasar parámetros definidos en forma dinámica. Dado que esto último no es posible hacerlo aún en GeneXus, pero existe la posibilidad que se implemente a futuro, se resolvió construir un prototipo que genere pruebas solo para procedimientos con dos parámetros de entrada y uno de salida.
- El objeto verificador se implementa como un WS que es consumido desde la extensión.
- Crea un objeto nuevo en la KB denominado "*TestSet*": Dicho objeto permite definir los casos de prueba para un objeto a verificar. Tiene un editor que permite ingresar las "tuplas o casos de

[12] eXtensible Markup Language

[13] Los objetos GeneXus se componen de partes, donde se especifican diferentes aspectos de los mismos. Cada parte tiene su editor. Son ejemplos de partes de objetos las siguientes: código fuente, código de las reglas, código de eventos, formularios, diseño de salida impresa

prueba" con datos de entrada y salidas esperadas, así como los procedimientos verificables y comentarios.

- Almacena los datos para pruebas en la base de datos en que reposa la KB, mientras que los resultados son almacenados en archivos XML.
- Permite reconstruir un caso de prueba ante cambios en la regla *parm* del procedimiento a verificar minimizando la pérdida de datos.
- Genera una bitácora ("*log*") con diferentes niveles de detalle.

3.4 Implementaciones en acción

Se resumirá a continuación información recogida de los manuales de usuario escritos por los grupos de estudiantes, a efectos de ejemplificar acerca de la utilización y el alcance de lo producido.

3.4.1 GxUnit1

Se desea crear un objeto para verificar un *procedure* (procedimiento) GeneXus. En las figuras 2 y 3 se muestran sus parámetros (regla *parm*) y su código, respectivamente.

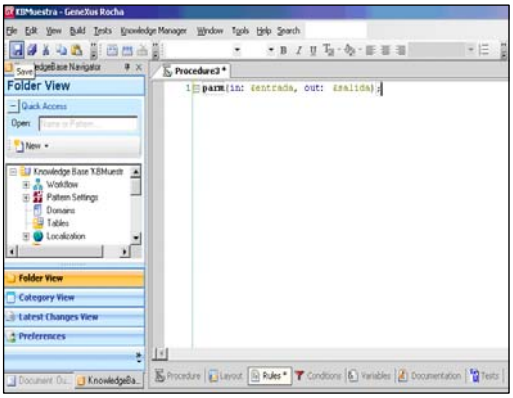

Figura 2: Regla "*parm*" de procedimiento a probar.

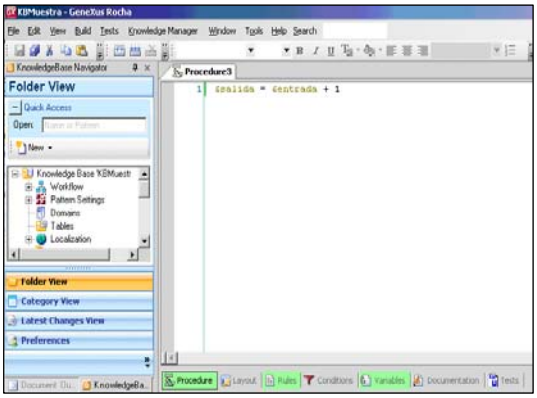

Figura 3: Código fuente a verificar.

Los datos de prueba se ingresan a la pestaña "*Tests*" (parte agregada al objeto). El editor implementado muestra la grilla, donde las columnas se corresponden a los parámetros del procedimiento a probar y las filas a los casos de prueba. Es posible seleccionar qué casos de prueba se van a correr en la próxima ejecución. También es posible cargar la grilla a partir de un archivo XML (marcando la opción "*Import from…*"). Ver figura 4.

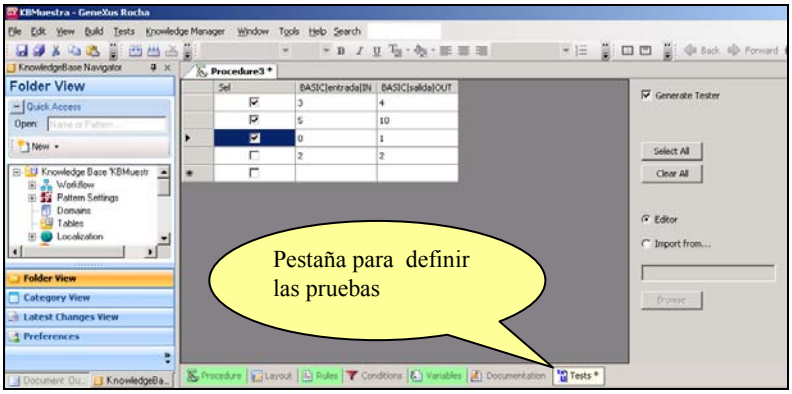

Figura 4: Editor de pruebas.

Al guardar el procedimiento a probar se genera el procedimiento verificador y el archivo XML con los datos de los casos de prueba. Ver figura 5.

Figura 5: Creación del procedimiento verificador.

Se ejecutan con la opción "*Run Tests*" del menú "*Tests*". Ver figuras 6 y 7.

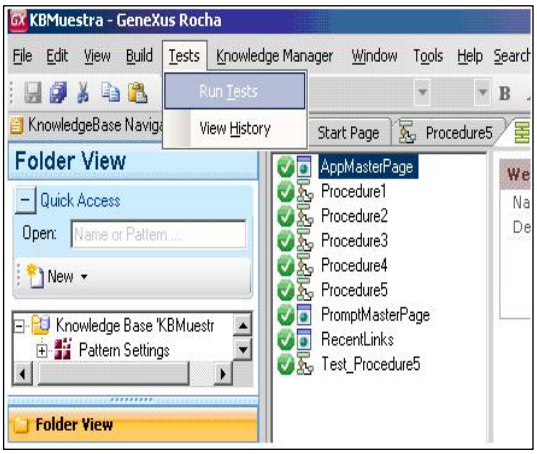

Figura 6: Menú para ejecutar pruebas.

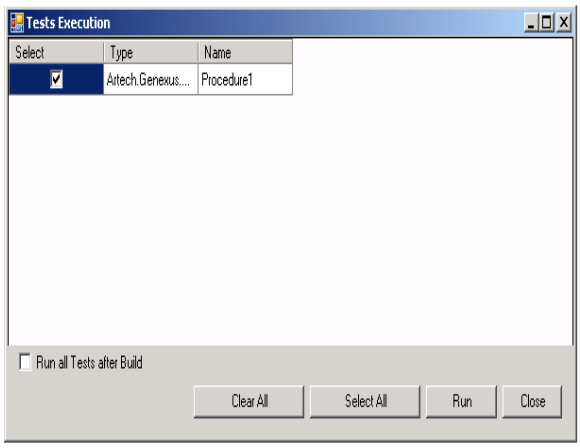

Figura 7: Selección de pruebas a ejecutar.

El resultado de la ejecución aparecerá en un archivo XML, visible desde el IDE.

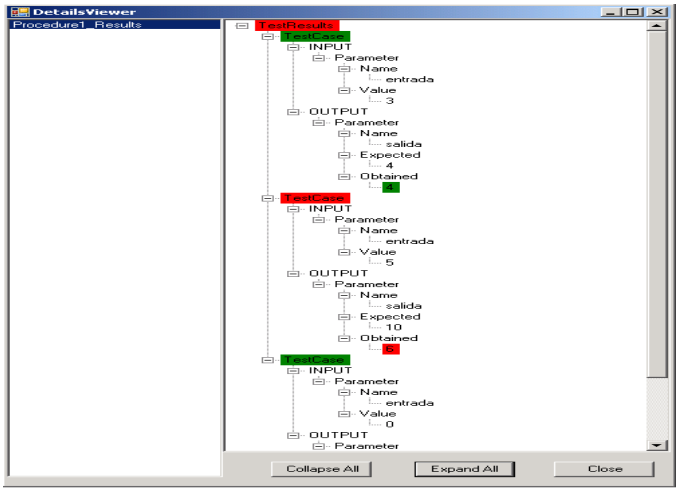

Figura 8: Vista expandida de los resultados.

3.4.2 GxUnit2

Se desea verificar un *procedure* GeneXus. Para esto se debe crear un objeto del tipo "*TestSet*", tal como se muestra en la figura 9.

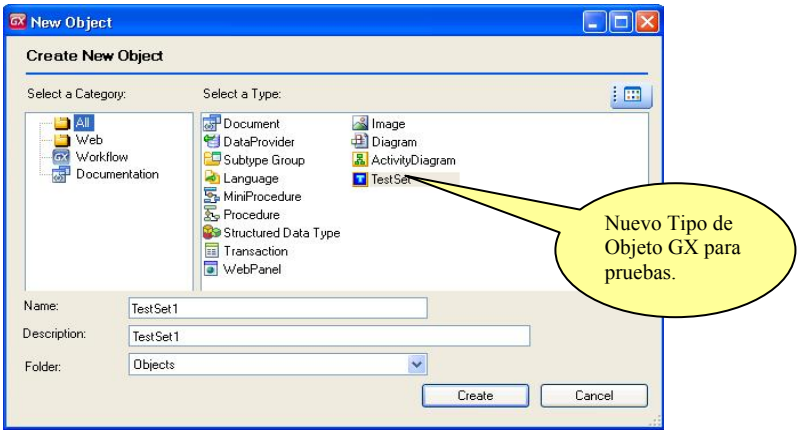

Figura 9: Creación de objeto *TestSet*.

Luego de creado se presenta una pantalla para definir la prueba. Se solicitará el nombre del procedimiento verificable a probar, los casos de prueba y los PVUs. En el ejemplo se creará una instancia de un objeto *TestSet* llamado TestSetSumador cuyo objetivo es probar un procedimiento verificable existente en la KB llamado "Sumador". El editor de dicho objeto se muestra en la figura 10. El procedimiento Sumador recibe como entradas dos parámetros y ofrece como salida otro parámetro (de igual tipo y dimensión) con la suma de los valores de los parámetros de entrada. Un campo de ingreso tipo "combo de selección" permite ubicarlo y seleccionarlo. El editor despliega los parámetros del procedimiento en una grilla. El Analista puede especificar cuales parámetros interesan para la prueba. Luego se le presentará una grilla donde se podrán ingresar todos los casos de prueba. Las columnas de la grilla corresponderán a cada uno de los parámetros involucrados y cada fila corresponderá a un nuevo caso de prueba. El procedimiento a probar se ejecutará una vez por cada fila de la grilla cuando se dispare la prueba. Finalmente, es posible indicar uno o más PVUs para verificaciones adicionales. Se proponen al Analista como candidatos a seleccionar desde un combo, los procedimientos que tienen igual número y tipo de parámetros que el procedimiento a verificar y uno adicional, como salida, de tipo "*booleano*". Es posible indicar si se deben ejecutar o no los PVUs seleccionados.

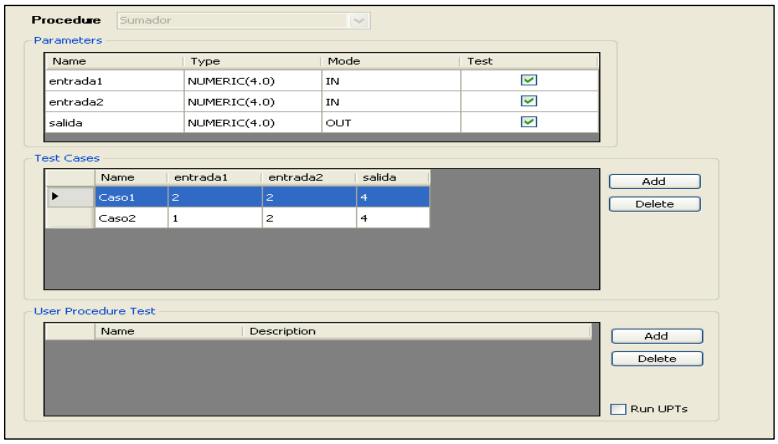

Figura 10: Editor de *TestSet* y sus grillas para ingreso de casos de prueba.

Para ejecutar las pruebas se utiliza el menú "*Tool Windows"* del IDE de GeneXus, al que se le agregó la opción "*Testeable Procedures*". Ver figuras 11 y 12.

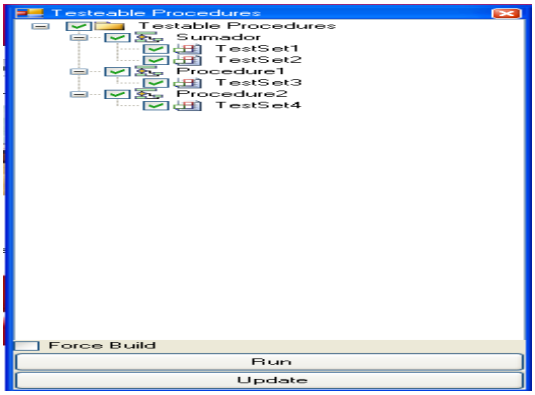

Figura 11: Menú para ejecución de pruebas.

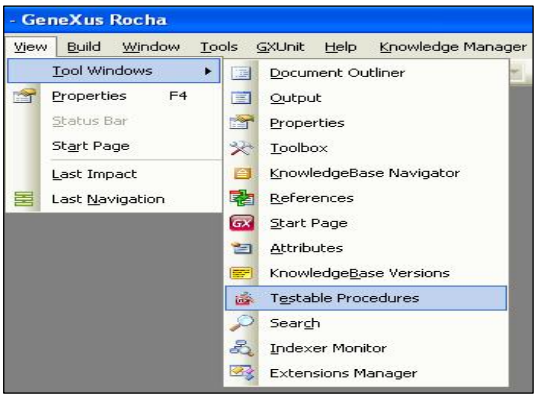

Figura 12: Pruebas a ejecutar.

El resultado general de la ejecución se expone tal como muestra la figura 13.

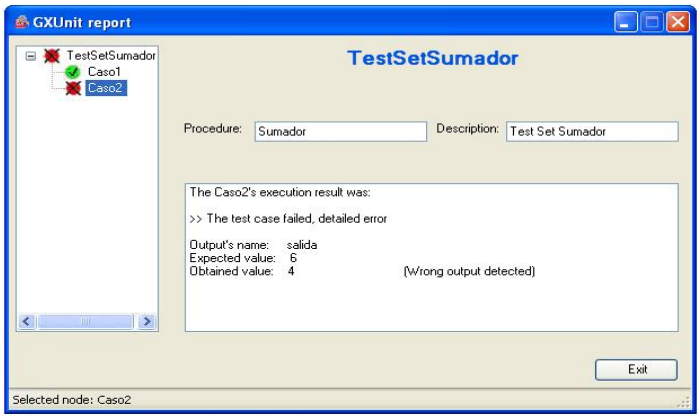

Figura 13: Reporte de ejecución (GxUnit2).

Además de presentar el reporte de resultados, la extensión registra detalles de la ejecución en un archivo XML según el nivel de detalle indicado por el Analista. La pantalla del visor de dicha bitácora ("*log"*) se expone en la figura 14.

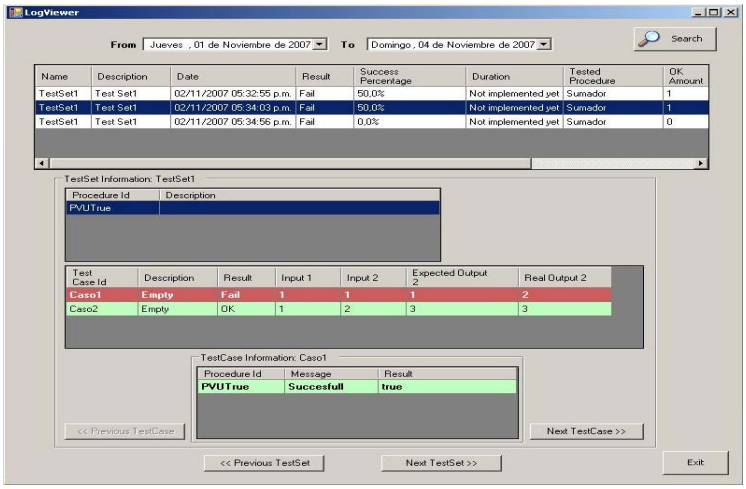

Figura 14: Visor de la bitácora.

Los objetos *TestSet* son capaces de responder frente a una modificación externa de los procedimientos a verificar o de los PVUs. Tras modificar los parámetros de un procedimiento a verificar, al abrir el *TestSet* que lo verifica presenta una ventana similar a la mostrada en la figura 15. Observando la figura se aprecia que en el cuadrante superior izquierdo de la pantalla se presentan los parámetros que tenía el procedimiento a verificar previo el cambio y debajo los nuevos parámetros. En el cuadrante superior derecho se muestran los antiguos casos de prueba, que perdieron validez tras el cambio. Debajo se pueden agregar casos de prueba de forma que contemplen los cambios. Los antiguos valores pueden ser reutilizados, ya sea escribiéndolos manualmente o arrastrándolos desde una grilla a otra.

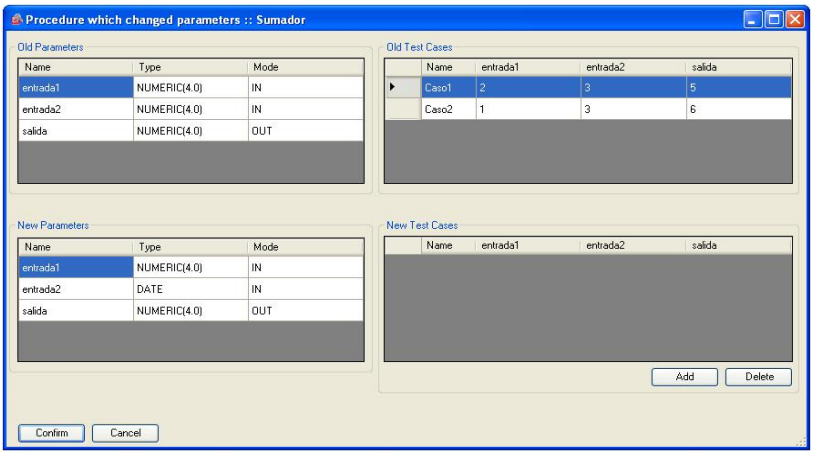

Figura 15: Impacto en el *TestSet* por cambios en la regla *parm*.

4. CONCLUSIONES Y TRABAJO FUTURO

Se han logrado desarrollar los primeros prototipos de una herramienta de pruebas unitarias para ambientes de desarrollo en GeneXus, de forma colaborativa y *open-source*. La forma de continuar puede tomar varios caminos, siguiendo las diferentes vertientes referidas a la especificación, desarrollo y aplicación. En cuanto a la especificación, se deberá ampliar el alcance con el objetivo de abarcar los objetos GeneXus aún no contemplados, especialmente aquellos con interfase *web* y *win*. En relación al desarrollo, el primer camino está dado por el desarrollo de los prototipos de GxUnit por parte de futuros equipos externos a Artech hasta transformarlos en herramientas que otorguen valor para los Analistas. El segundo camino está determinado por el desarrollo de funcionalidad para pruebas unitarias que la propia empresa Artech estime pertinente realizar en GeneXus. En cuanto a su aplicación se entiende necesario realizar varios estudios de campo, una vez que la herramienta alcance un nivel de madurez que los permita.

5. REFERENCIAS

[1] Almeida E. “En el proceso de crear un *framework* de *testing*”,2003, http://ealmeida.blogspot.com/2003/10/estoy-en-el-proceso-de-crear-un.html,

[2] Almeida E. “Software Testing: Tres enfoques para un mismo problema”, Encuentro Internacional de Usuarios GeneXus del año 2004.

[3] Almeida E.,Araújo A.,Larre Borges U. “Proyecto colaborativo GxUnit”,2006, http://www.gxopen.com/commwiki/servlet/hwiki?GxUnit,

[4] Almeida E. ,Araújo A.,Larre Borges U. “Collaborative Projects”, 2006 http://www.concepto.com.uy/archivosvinculados/EncuentroGX2006CollaborativeProjects.ppt

[5] Artech Consultores S.R.L. “Visión general de GeneXus”,2005, http://www.GeneXus.com/portal/agxppdwn.aspx?2,32,660,O,S,0,22790%3bS%3b1%3b2315,

[6] Cunningham W. “Framework for Integrated Tests”, 2007, http://fit.c2.com/,
[7] GeneXus, http://www.GeneXus.com, 2008.
[8] GeneXus Community Wiki, “GeneXus Rocha/GXextensions”, 2007, http://wiki.gxtechnical.com/commwiki/servlet/hwiki?category%3AGeneXus+Extensions
[9] Gonda B.,Jodal N. “Filosofía y Fundamentos Teóricos de GeneXus”,2007, http://www.GeneXus.com/portal/hgxpp001.aspx?2,32,660,O,S,0,MNU;E;131;12;MNU.
[10] Gonda B.,Jodal N. “Proyecto GeneXus”. Academia Nacional de Ingeniería, Uruguay, 1995 http://www.aiu.org.uy/gxpsites/agxppdwn?2,1,4,O,S,0,79%3BS%3B1%3B21.
[11] GxUnit1, http://www.gxopen.com/gxopenrocha/servlet/hproject?721, 2008.
[12] GxUnit2, http://www.gxopen.com/gxopenrocha/servlet/hproject?721, 2008.
[13] Janicki R.,Parnas D., Zucker J., “Tabular Representations in Relational Documents”, Communications Research Laboratory, McMaster University, 1996.
[14] JUnit, http://junit.org, 2008.
[15] Proyecto de Ingeniería de Software 2007 “GxUnit”, Docente: Triñanes J. ,Facultad de Ingeniería, Universidad de la República, Uruguay, 2007.
[16] Melnik G. “Empirical Analyses of Executable Acceptance Test Driven Development”, PHD Th., Supervisor: Maurer, F., DCS, University of Calgary, Calgary, Alberta, http://ebe.cpsc.ucalgary.ca/ebe/uploads/Publications/MelnikPhD.pdf, 2007.
[17] Melnik G.,Maurer F. “Multiple Perspectives on Executable Acceptance Test-Driven Development”, DCS, University of Calgary, Canada, 2007, http://ebe.cpsc.ucalgary.ca/ebe/uploads/Publications/XP2007_Melnik_Maurer.pdf,
[18] Melnik G.,Read K., Maurer, F. “Suitability of FIT User Acceptance Tests for Specifying Functional Requirements: Developer Perspective”, DCS, University of Calgary, Canada, 2004 http://ebe.cpsc.ucalgary.ca/ebe/uploads/Publications/MelnikReadMaurer2004b.pdf,
[19] Miller J. “Is there a future for Fit testing?”, 2007 http://codebetter.com/blogs/jeremy.miller/archive/2007/07/30/is-there-a-future-for-fit-testing.aspx
[20] Mugridge R., Cunningham W. ”Fit for Developing Software: Framework for Integrated Tests”, ISBN 0-321-26934-9, Prentice Hall PTR, 2005.
[21] Latorres E.,Salvetto P., Larre Borges U., Nogueira, J., “Una herramienta de apoyo a la gestión del proceso de desarrollo de software”, IX Congreso Argentino de Ciencias de la Computación (CACIC 2003), La Plata, Argentina.
[22] Read K. “Supporting Agile Teams of Teams via Test Driven Design”, Master Th, DCS, University of Calgary, Calgary, Canada, 2005 http://ebe.cpsc.ucalgary.ca/ebe/uploads/Publications/Read2005.pdf, 2005.
[23] Salvetto P. “Modelos Automatizables de Estimación muy Temprana del Tiempo y Esfuerzo de Desarrollo de Sistemas de Información”, Tesis de Doctorado, Directores: Segovia, F., Nogueira J., Fac.Informática, Univ.Politécnica de Madrid, Doctorado conjunto en ingeniería informática UPM-ORT, 2006 http://oa.upm.es/367/01/PEDRO_SALVETTO_LEON.pdf,
[24] Shores J. “James Shores Successfull Software” “Five Ways to Misuse FIT”, 2007 http://www.jamesshore.com/Blog/Five-Ways-to-Misuse-Fit.html, 2007.

Agradecimientos

A los integrantes del equipo que desarrolló GxUnit1: Adrián García, Anthony Figueroa, Antonio Malaquina, Cecilia Apa, Darío de León, Diego Gawenda, Diego San Esteban, Federico Parins, Fernando Colman, Ken Tenzer, Lucía Adinolfi, Marcelo Falcón, Rafael Sisto, Rodrigo Ordeix.

A los integrantes del equipo que desarrolló GxUnit2: Fernando Varesi, Gervasio Marchand, Guillermo Pérez, Guillermo Polito, Horacio López, Ignacio Esmite, Marcelo Celio, Marcelo Vignolo, Martín Sellanes, Nicolás Alvarez de Ron, Rodrigo Aguerre, Rosana Robaina, Soledad Pérez, Stephanie De León.