Instituto de Computación – Facultad de Ingeniería – Universidad de la República

TESIS DE GRADO
INGENIERÍA EN COMPUTACIÓN

GENERACIÓN AUTOMÁTICA DE CODECS PARA TTCN-3 / JAVA

Extensiones Java para el T3DevKit sobre TTworkbench

Hernán Martínez

Supervisor: Ariel Sabiguero Yawelak

Tribunal: Daniel Calegari
Héctor Cancela
Mónica Wodzislawski

Resumen

TTCN-3 es un lenguaje estandarizado, enfocado al diseño de casos de prueba para distintos productos y sistemas. Para ejecutar un caso de test, es necesario contar con una especificación abstracta TTCN-3, pero además, se debe contar con un conjunto de entidades que permiten realizar el acceso al sistema bajo prueba. TTCN-3 permite definir pruebas de forma sencilla y realizar un seguimiento detallado de su ejecución.

El T3DevKit es una herramienta que ayuda al desarrollo de las piezas de software necesarias para ejecutar una especificación abstracta en lenguaje TTCN-3. Brinda una biblioteca con la implementación de las interfaces estándar del lenguaje basada en clases C++, que permite manipular fácilmente datos TTCN-3 con diferentes niveles de abstracción, mecanismos para los métodos de codificación/de-codificación e implementa la comunicación con puertos, temporizadores y funciones externas. La mayoría del código es generado por el CoDec Generator automáticamente, reduciendo la cantidad de código a ser escrito por el usuario, ayudando a aumentar la productividad y obteniéndose código más fácil de mantener.

Este proyecto tiene como objetivo la generación de una pieza de software que permite adaptar el T3DevKit para ser utilizado en Java. Para la realización del mismo se contó con el compilador TTCN-3 denominado Testing Tech Workbench sobre la plataforma Java. Durante este trabajo lo integramos con el T3DevKit, permitiendo ejecutar casos de prueba concretos.

Al contar con módulos Java y C/C++ a ser integrados e interoperar, se optó por utilizar la tecnología JNI para conectarlos. Lo que da lugar a lo que llamamos indistintamente conector o wrapper JNI. Esta pieza de software es la encargada de realizar las correspondencias entre funciones y tipos declarados en ambos lenguajes, permitiendo invocaciones desde Java hacia C/C++ y viceversa.

Palabras clave: Generación automática de CoDecs, Java, TTCN-3, TTwb, T3DevKit.

Índice de contenido

Indices

Índice de ilustraciones

Índice de tablas

1. Introducción

Este trabajo se basa en el lenguaje TTCN-3 (Testing and Test Control Notation version 3) y las herramientas relacionadas con esta tecnología. TTCN-3 es un lenguaje cuyos objetivos principales son la especificación y ejecución de casos de prueba, aplicados a diferentes áreas, equipos y sistemas. Se trata de un lenguaje de testing estandarizado por ETSI [1], para la ejecución de un Test Case, no sólo alcanza con interpretar y compilar sus fuentes, sino que también, es necesario contar con un conjunto de entidades que permiten implementar la interacción especifica con el SUT (System Under Test o Sistema Bajo Pruebas). El lenguaje TTCN-3 está definido, en su última versión, en los estándares que aparecen en las referencias: [2], [3], [4], [5], [6] y [7]. Esta tecnología busca la descripción simple, abstracta y sin ambigüedad de pruebas y su comportamiento, indicando con claridad si una prueba es exitosa o no.

En el contexto de la realización de pruebas, el objetivo final es proveer un ETS (Executable Test Suites) que se corre contra un IUT (Implementation Under Test sinónimo de SUT en este documento). Se han realizados diferentes trabajos con el objetivo de proporcionar lenguajes que especifiquen ATS (Abstract Test Suites) y entornos para obtener un ETS, dentro de los cuales está TTCN-3.

El problema que se presenta, es que los lenguajes a ser utilizados para la definición de un ATS deben ser lo más abstractos posibles, para facilitar la especificación de escenarios para las pruebas. Esta abstracción, ayuda a la portabilidad y re-utilización de definiciones para las pruebas. Por el contrario, los entornos utilizados para ejecutar un ETS contra un IUT, están diseñados para ser lo más cercanos posible a las características concretas de estas implementaciones. Por esta razón se puede observar que existe una brecha entre ATS y ETS, el trabajo para obtener un ETS a partir de un ATS, sigue siendo un procedimiento complejo y propenso a errores. Esto motiva a que los desarrolladores de las pruebas intenten escribir directamente el ATS lo más ajustado posible al ETS y utilizando el mismo bajo nivel de lenguajes de programación que se ha utilizado para el desarrollo de la aplicación a ser probada. Esto tiene como inconvenientes que el ETS obtenido puede no corresponderse con los tests propuestos previamente en un nivel más abstracto, mientras que para ser ejecutado en otra aplicación o en otro entorno, las pruebas obtenidas podrían tener que ser reescritas por completo.

El crecimiento en la complejidad de los sistemas requiere de más y mejores pruebas, por lo que es necesario buscar formas de reducir la brecha entre ATS y ETS,

permitiendo la especificación del ATS en un lenguaje abstracto de alto nivel y obteniendo el ETS de manera automática, o al menos disminuyendo el trabajo manual. TTCN-3 intenta aportar una solución a esta problemática, pero todavía hay mucho trabajo ha ser realizado sobre esta tecnología, para proporcionar las herramientas y entornos necesarios.

El proceso de construir el conjunto de entidades requeridas por TTCN-3 durante el pasaje de ATS a ETS es costoso. Una de las motivaciones centrales de nuestro proyecto, fue generar una pieza de software que permita la re-utilización de componentes existentes. Otros objetivos buscados fueron además, fomentar la interoperabilidad entre las plataformas Java y C++ y además conseguir un buen nivel en la mantenibilidad de los productos.

El T3DevKit no es una herramienta completamente independiente de la implementación de las interfaces que brinde el compilador TTCN-3 que se esté utilizando, por lo que, se incluyó como objetivo un análisis de los impactos generados por el uso del compilador Java mencionado; esto llevó a que se realizaran distintas adaptaciones y complementos a la herramienta para lograr la integración adecuada sin afectar las funcionalidades ya disponibles.

Otro punto interesante para cumplir con la construcción del conector, es la revisión de la implementación de tipos TTCN-3, esto aplica tanto a los objetos utilizados por el compilador Java, así como también, a las definiciones de los tipos C/C++ definidos en la interfaces TCI Value.

Se cuenta con el compilador Java TestingTech [8] y la herramienta para TTCN-3 CoDec Generator C/C++ de IRISA [9], la solución diseñada consta de un conector JNI [10], que mediante la implementación nativa de las interfaces definidas por Go4IT [11] y basadas en los estándares de ETSI [1]; más la utilización de las funcionalidades incluidas en el T3DevKit, brinda los componentes genéricos Adapter y CoDec necesarios; estos que permiten la ejecución de casos de test escritos en lenguaje TTCN-3.

La responsabilidad del conector JNI es la de implementar de forma nativa las interfaces Java Provided (para TCI/CD [12], TRI/SA y TRI/PA [13]) re-utilizando las funcionalidades brindadas por el T3DevKit y luego permitir la invocación desde C/C++ a las interfaces Required (para TCI/CD, TRI/SA TRI/PA) implementadas en Java, estas son utilizadas para generar los objectos de transferencia desde el T3DevKit.

Desde un principio, se trazaron como objetivos de diseño e implementación, que la solución tuviera un alto grado de mantenibilidad, buscando simplicidad en las entidades y módulos que la componen. Otro punto importante es que al basarse en interfaces ya definidas y la gran mayoría estandarizadas, finalmente se obtuvo un producto que conecta a ambas plataformas, pero las mismas permanecen independientes, por lo que son capaces de ser mantenidas en forma separada sin tener impacto en la implementación de la otra.

Para validar el diseño y la implementación del conector, se generaron casos de test escritos en TTCN-3. Inicialmente sólo se probó la trasferencia de TCI Values, para después integrar el CoDec con implementación nativa. Luego de pasadas la pruebas anteriores, se paso a la ejecución con el Adapter totalmente implementado en forma

nativa, para ello se utilizó el conjunto de Test Cases incluidos en el T3DevKit, detectándose problemas a la hora de invocar funciones externas, dados por la utilización no estandarizada de algunas interfaces por parte del TTwb.

El documento está organizado de la siguiente forma: en la Sección 1. se encuentra la introducción con un breve recorrido por los diferentes aspectos del proyecto; en la Sección 2. se realiza un repaso del estado de la tecnologías a ser utilizadas e integradas (T3DevKit), así como una introducción al lenguaje TTCN-3, su arquitectura y las entidades participantes en la ejecución de un Teste Case; en la Sección 3. se realiza una descripción del problema a resolver con su planteo inicial, sus requerimientos, los detalles sobre las interfaces y sus especificaciones, se incluye un resumen sobre lo mencionado a lo largo de esta sección; en la Sección 4. se comienza a describir el diseño del conector, como interactúan las entidades CoDec y Adapter, como se realizan las invocaciones a las interfaces Provided y como se agrupan cierta funcionalidades comunes a los módulos contenidos en el conector; en la Sección 5. se dan los detalles de implementación de los módulos, marcando las correspondencias entre las declaraciones de la interfaces en Java y C/C++, como están declaradas las estructuras que representan los TCI Value, se detalla la implementación de la conexiones Java → C y viceversa, dando lugar a la descripción de funcionalidades compartidas por los módulos y como se maneja la configuración al ejecutarse un Test Case; en la Sección 6. se presentan las pruebas realizadas, con respecto a la transferencia de TCI Values, la integración del CoDec y el Adapter, para luego analizar los resultados obtenidos; en la Sección 7. se mencionan algunos trabajos y participaciones vinculadas a este proyecto; en la Sección 8. se desarrollan las conclusiones obtenidas; en la Sección 9. se incluye el glosario con las abreviaturas y conceptos utilizados a lo largo del documento.

Los apéndices tienen el siguiente contenido: en el Apéndice A. se incluye un documento, donde se indica como realizar la instalación del TTwb, habilitar la licencia y realizar la configuración del entorno de desarrollo; en el ApéndiceError: No se encuentra la fuente de referencia se incluyen las instrucciones para la instalación de los módulos componentes del conector; en el Apéndice C. se muestra el seguimiento de la ejecución de pruebas para TCI Values; en el Apéndice D. aparece el seguimiento de la ejecución de pruebas para el módulo CoDec; en el Apéndice E. se ubica el seguimiento de la ejecución de pruebas con los módulos CoDec y Adapter en funcionamiento; en el Apéndice F. se muestra el seguimiento de la ejecución para el test RIPng, el cual realiza las invocaciones a funciones externas; en el Apéndice G. se incluye la carta enviada a la organización de la T3UC 2008.

2. Estado del Arte

En esta sección se presentan los conceptos, definiciones y herramientas utilizados a lo largo de este trabajo. Comenzando con el lenguaje TTCN-3, la herramienta de desarrollo automático de CoDecs T3DevKit y el ambiente integrado de desarrollo TTworkbench.

2.1. El lenguaje TTCN-3

El Testing and Test Control Notation version 3 (TTCN-3) [14], es un lenguaje de testing normalizado para la definición de especificaciones abstractas de casos de prueba en una amplia gama de equipos y sistemas. Está internacionalmente estandarizado y promovido por la European Telecommunications Standards Institute (ETSI).

Permite la descripción concisa de las pruebas y el comportamiento definiendo claramente el significado de si una prueba pasa, falla o queda indefinida. En estos momentos se considera un lenguaje universal de pruebas cuya área de aplicación no está sólo restringida a las pruebas de los sistemas de telecomunicaciones.

2.1.1. Características

TTCN-3 es un lenguaje de programación de propósito general, fuertemente tipado y procedural, originado en el dominio de soluciones para las telecomunicaciones, aunque el campo de aplicación ha ido creciendo con el tiempo y ha sido utilizado exitosamente en casi cualquier ámbito donde deben realizarse pruebas, estos van desde los Servicios Web a sistemas de tiempo real.

En la evolución del lenguaje la abreviatura (TTCN) se mantuvo, pero con un gran cambio de la tecnología subyacente. Después del re-diseño del lenguaje, se mantienen características de TTCN-2, aunque la notación fue cambiada drásticamente y la nueva "cara" es la de un moderno lenguaje de programación. Notaciones arborescentes y tabulares fueron sustituidas, convirtiéndose en representaciones gráficas que pueden ser traducidas dentro del lenguaje TTCN-3. La notación tabular y Message Sequence Charts (MSC), tienen estandarizada la traducción para y desde la nueva versión del lenguaje [14]. Otros avances se han intentado introducir, como por ejemplo, la orientación a

objetos, aunque hay cierta semejanza con el enfoque de orientación a objetos en algunas operaciones, finalmente la orientación a objetos fue dejada de lado en la notación.

TTCN-3 fue diseñado específicamente para la realización de pruebas, si bien muchas características son similares a los de otros lenguajes de programación, contiene operaciones específicas para la realización de pruebas que lo distinguen claramente de otros lenguajes. Ha sido ampliado con nuevos conceptos no disponibles en otras versiones del lenguaje, estos conceptos incluyen el manejo de correspondencia de datos, la posibilidad de realizar pruebas distribuidas de acuerdo a la arquitectura del sistema a testear, y la ejecución concurrente de casos de prueba.

Contiene un sistema de tipo más amplio y variado de lo usual comparándolo con otros lenguajes de programación tradicionales, incluye tipos nativos de listas, un tipo específico para los veredictos, y tipos de datos para representación de timers. Además TTCN-3 proporciona soporte directo sobre temporizadores, así como para el envío y recepción de mensajes, brindando funcionalidades que llevan a cabo los mecanismos de comunicación.

Dentro del los fuentes, es muy claro el significado de cada elemento del lenguaje, cada uno de los componentes del mismo son especificados con claridad y precisión. Esto indica que una secuencia de comandos de prueba por escrito en TTCN-3 es inequívoca. Esta precisa definición del lenguaje, también lleva a que la herramienta tome independencia de sus proveedores, ya que cada herramienta debe ejecutar un caso de prueba exactamente de la misma manera. Que la herramienta sea independiente del vendedor facilita el movimiento de un compilador TTCN-3 a otro conjunto de herramientas y ayuda en gran medida en proyectos en los que las pruebas se ejecuten con herramientas de diferentes fabricantes utilizadas en paralelo.

TTCN-3 está diseñado para brindar un lenguaje de testing simple y de propósito general a la hora de implementar las pruebas, siendo adecuado para una amplia gama de aplicaciones de test. Se puede utilizar a lo largo de todo el ciclo de desarrollo de un producto. De esta manera, puede ofrecer importantes beneficios en términos de rendimiento al momento de la inversión en herramientas de testeo, la formación, y por supuesto, la calidad de los productos.

En su corazón, TTCN-3 tiene un poderoso e intuitivo formato de texto para la definición de la pruebas y escenarios, que es similar al procedimiento convencional de los lenguajes de programación, este formato se refiere como la notación básica de TTCN-3 [15].

2.1.2. Arquitectura de TTCN-3

La distribución e interacción de los componentes de TTCN-3 se muestra en la Figura 2.1 y a continuación se enumeran las características principales de la arquitectura [15]:

- Un sistema de test (Tester) debe ser interpretado como un conjunto de entidades que interactúan y pueden ser desarrolladas en forma paralela.
- Solamente el TTCN-3 Executable (TE) es escrito en el lenguaje TTCN-3 y si es necesario, es posible incluir las llamadas a funciones externas implementadas en otro lenguaje.

- Los demás componentes están escritos en un lenguaje de plataforma base (Java o C/C++), o sea la implementación de las interfaces de comunicación (TRI/SA, adaptador con el sistema bajo testeo), interfaces para el manejo de temporizadores (TRI/PA, adaptador con la plataforma) y módulo de codificación y de-codificación (TCI/CD, implementación de codificación y de-codificación de tipos y mensajes).

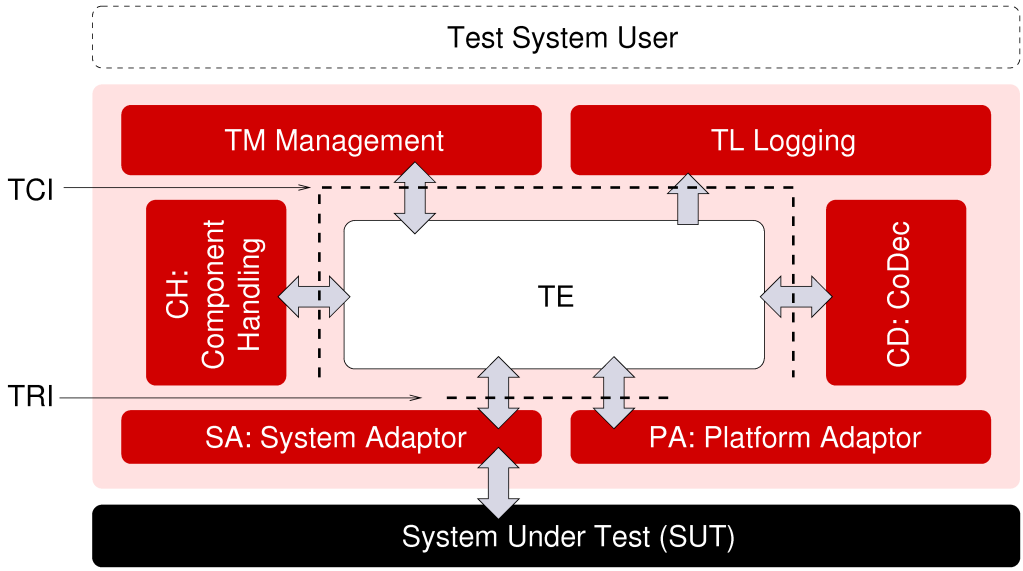


Figura 2.1: Arquitectura de TTCN-3

El concepto de Implementation Under Test (IUT) es conocido como System Under Test (SUT) en los estándares del lenguaje. El elemento central en la figura Figura 2.1 es el TE, encargado de ejecutar los módulos TTCN-3 compilados, tiene el control sobre los Test Case, componentes, valores y elementos en las estructuras definidas en los fuentes TTCN-3, este puede ser ejecutado de manera centralizada o distribuida.

A pesar de la abstracción del lenguaje, las operaciones deben ser aplicadas a bajo nivel de algún modo. La implementación del TE es una descripción abstracta de un módulo TTCN-3 y de otras entidades incluidas por los estándares, estas entidades y sus interacciones brindan la implementación específica para la ejecución de las pruebas. Por ejemplo, el concepto abstracto del envío de un mensaje (operación `send()`) , debe ser complementado con las entidades que indican la manera de codificar el mismo y enviarlo por un medio físico concreto. La API que complementa la especificación abstracta de un test, está dada por las interfaces estandarizadas: TTCN-3 Runtime Interface (TRI) y TTCN-3 Control Interface (TCI).

La TTCN-3 Runtime Interface define la interacción entre el TE, el System Adaptor (SA) y el Platform Adaptor (PA) [13]. Proporciona los medios para que el TE pueda enviar los datos hacia el SUT, recibir respuestas y/o manejar temporizadores, entre otras tareas. El TRI está dividido en dos sub-interfaces bidireccionales: triCommunication y triPlatform. La interfaz triCommunication refiere a la comunicación con el SUT, que

será implementada en bajo nivel dentro del SA. La interfaz triPlatform, permite la correspondencia de un ETS particular, a una plataforma de ejecución específica.

La TTCN-3 Control Interface (TCI) define la interacción entre el TE y el Test Management and Control (TMC) [12]. La entidad TMC incluye las funcionalidades relativas al manejo de la ejecución de un test, componentes, codificación y de-codificación de datos intercambiados con el SUT. Las entidades Test Management (TM), Coding and Decoding (CD) y Component Handling (CH) constituyen el TMC. El TM es responsable de toda la gestión de un Test System, realizando las invocaciones apropiadas de los módulos TTCN-3, incluyendo la configuración de los parámetros del test en la ejecución actual. El CD es la entidad encargada de brindar las transformaciones entre las representaciones de los mensajes TTCN-3 a cadenas de bits transferibles y viceversa.

El TE determinara que CoDecs serán utilizados, de esta manera, los datos TTCN-3 son manejados por una rutina apropiada para obtener la representación orientada a bits (representación transmisible de los datos); cuando los datos son recibidos por el SUT, estos son decodificados por la entidad CD y convertidos a valores TTCN-3. La entidad CH maneja la ejecución de Test Cases en paralelo y como el TE puede ser distribuido en diferentes dispositivos, la CH provee servicios para sincronizar las entidades del Test System.

La restante entidad incluida en TCI es el Test Logging (TL). Este realiza la tareas referentes al manejo de la información a incluir en el log, proveniente de las otras entidades a partir de sus eventos de logging y la presentación del Test System al usuario. Por ejemplo, la creación de componentes dentro del test, la información intercambiada (enviada o recibida) con el SUT, correspondencias entre instancias de valores TTCN-3 y operaciones de temporización son tomadas por estas funciones de seguimiento.

Las interfaces incluidas en TCI y TRI son clasificadas en Provided y Required. Las Provided deben ser implementadas por el usuario, permitiendo que el TE las invoque a la hora de la comunicación con el SUT; mientras que, las Required son implementadas dentro del TE y permiten que el SUT haga efectiva la comunicación inversa (realizando las peticiones y respuestas al TE).

2.1.3. Análisis detallado del TE

En el contexto del proyecto Go4IT [11] se avanza aún más en la descripción del TE (TTCN-3 Executable). El compilador TTCN-3 y las librerías de Runtime asociadas son los módulos principales de software de la plataforma de desarrollo.

Un compilador TTCN-3 típico, analiza el ATS sintácticamente y semánticamente, para traducir el código TTCN-3 a uno de los lenguajes de plataforma, obteniendo el Compiled ATS. Luego, el código generado para el lenguaje de plataforma, es compilado con un compilador estándar y es enlazado con las librerías TTCN-3 que implementan, entre otras, las funciones Required de las API TRI y TCI (TM-Mod, CH-Mod, CD-Mod, SA-Mod, PA-Mod).

La Figura 2.2 muestra en forma detalla como se compone el TE, y como se ubican los componentes descritos.

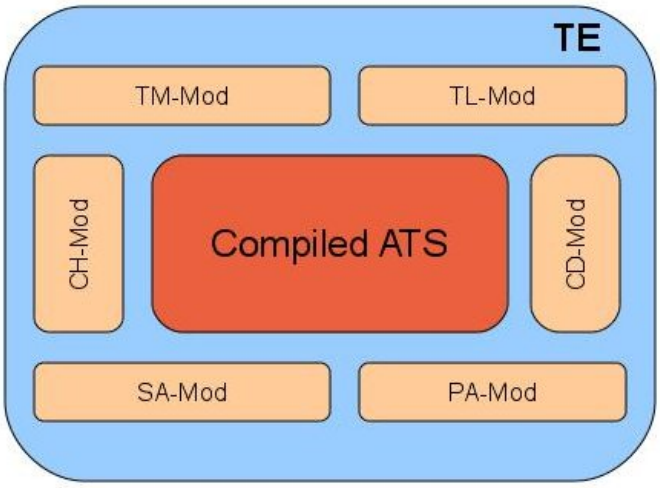


Figura 2.2: Detalle de la composición del TE.

2.2. T3DevKit

Es una herramienta que ayuda al desarrollo de las piezas de software necesarias para ejecutar un ATS (el concepto de Abstract Test Suite fue incorporado en el apartado 1.) escrito en lenguaje TTCN-3. A continuación se hace mención a sus características principales, funcionalidades mas destacadas, descripción del marco de trabajo que esta herramienta brinda y también sus limitaciones [9].

2.2.1. Características

La biblioteca brindada facilita el manejo de los valores TTCN-3 concretando su representación abstracta en C, esto ayuda al desarrollador a generar correctamente las estructuras utilizadas a través de las invocaciones a TCI/CD.

La mayoría del código es generado por el CoDec Generator automáticamente, por lo que se reduce al mínimo la cantidad de código a ser escrito a mano, ayudando a aumentar la productividad y obteniéndose código mas fácil de mantener.

El T3DevKit está compuesto por dos partes:

- T3DevLib es una biblioteca, que dispone de lo siguiente: una implementación de las interfaces estándar ETSI TRI y TCI/CD y un marco de trabajo basado en clases C++ para:
  - manipular fácilmente datos TTCN-3 con: diferentes niveles de abstracción, mecanismos para los métodos de codificación/de-codificación, manejo de excepciones a nivel de CoDec y funciones de depuración.
  - implementar la comunicación con puertos, temporizadores y funciones externas.

- T3CDGen procesa y extrae los tipos definidos en código fuente TTCN-3, generando la mayor parte del código C++ necesario para implementar el CoDec. El resultado CoDec es altamente personalizable y es posible escribir pequeñas porciones de código llamadas “Codets” para ello.

La herramienta no es completamente independiente del compilador por carencias de los estándares. Por lo que, se debe adaptar el T3DevKit para se comunique con un módulo intermedio y por lo tanto con TTwb; para lo que se utilizan la interfaces TCI y TRI publicadas en Go4IT. Al presente está desarrollado para los compiladores de Danet, Telelogic y Go4IT.

2.2.2. Marco de trabajo

La forma típica de integrar en un proyecto el T3DevKit se muestra en la Figura 2.3.

Para la compilación de un Abstract Test Suite (ATS) no al alcanza con traducir el código fuente TTCN-3. Por lo general, se tendrá que desarrollar un de CoDec y System Adapter para realizar la conexión entre lo "abstracto" y el "mundo real".

Los adaptadores están construidos encima del T3DevLib, que proporciona las interfaces de comunicación con los Test Executable y las funciones adicionales.

El CoDec se construye también encima del T3DevLib pero no se deben escribir manualmente. La mayor parte de su contenido, es generado automáticamente por el CoDec Generator T3CDGen, a partir de los fuentes de TTCN-3. Las únicas partes que deben de ser escritas manualmente son los Codets, los cuales se corresponden con algunas piezas de código C++, que son integrados al final del CoDec por T3CDGen.

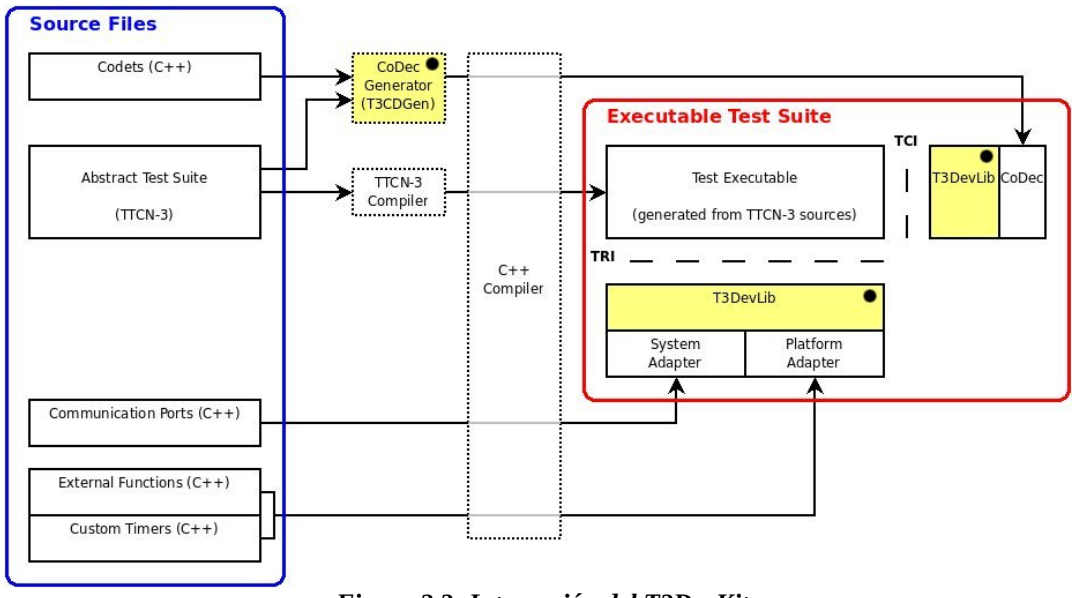


Figura 2.3: Integración del T3DevKit

En la Figura 2.3 están marcadas con un círculo negro, las partes provistas por el T3DevKit.

Cabe señalar, que en ningún momento es requerida la manipulación del código generado, todo está construido a partir de archivos fuente.

2.2.3. Necesidades de contar con una biblioteca y un CoDec Generator

El desarrollo y especialmente el mantenimiento de un CoDec es una tarea costosa. Lo mismo se aplica (en un grado inferior) al System y el Platform Adapter. Varias operaciones necesitan ser realizadas:

- **Sincronización de definiciones de tipos:** Las definiciones de tipos están escritas en TTCN-3, sin embargo en el interior del CoDec (escrito en C), el desarrollador necesita saber con precisión cómo los mensajes están representados en TTCN-3 con fin de instanciar correctamente las estructuras utilizadas a través de las invocaciones a TCI/CD. Cada definición de tipo TTCN-3 se corresponde con a una serie de llamadas a TCI/CD, básicamente el mismo trabajo se debe hacer dos veces. Se trata de una tarea tediosa y propensa a errores, sobre todo cuando las definiciones de tipo han sido actualizadas. Con el CoDec Generator la mayoría de código C++ ese generado automáticamente. La cantidad de código a ser escrito a mano se reduce al mínimo, por lo que es mucho más rápido de escribir y mucho más fácil de mantener.
- **Manipulación de datos TTCN-3:** Los valores TTCN-3 se representan de una manera abstracta en la TCI. Esto es dentro del lenguaje TTCN-3, pero no es fácil de manipular en C. Por ejemplo, un valor se recupera llamando `tciGetIntAbs()` que devuelve una representación ASCII del entero (por ejemplo, "42") y `tciGetIntSign()` que devuelve el signo. Esta no es una forma natural de manipular un entero en C. Algunos casting deben realizarse, y debe hacerse con cuidado (por problemas de desbordamientos, etc.). En la biblioteca, los enteros se representan con el estándar de enteros C, cadenas de caracteres con cadenas de caracteres C (puntero + longitud) y algunas funciones están previstas para facilitar el casting entre los tipos y la manipulación datos con bytes no alineados (como `bitstrings`).
- **Abstracción:** El desarrollador de los test y el desarrollador del CoDec no comparten la misma visión de los datos. El primero trabaja con valores abstractos y no se preocupan por la forma en que están codificados, mientras que el segundo manipula bits de datos codificados. La mayoría de las veces al desarrollador del CoDec no le importa si un campo representado de manera abstracta por un entero, un `charstring` o un `octetstring`, solamente ve un flujo de bits que tienen que ser divididos en campos. Sin embargo, si se trabaja directamente con invocaciones a TCI, tendrá que conocer el tipo exacto del campo, realizar el tipo de conversión adecuada y luego invocaciones a TCI para crear una instancia del valor correctamente. La biblioteca ofrece un entorno orientado a objetos que permite manipular cualquier variable TTCN-3 de un modo uniforme.

- **Multiplexado de invocaciones:** La mayoría de las llamadas a funciones TRI y TCI provistas por el usuario son multiplexadas. Por ejemplo, dos (o más) puertos de comunicación comparten las mismas primitivas: `TriMap()`, `TriUnmap()`, `TriSend()`, `TriCall()`, etc., Esto agrega una complejidad innecesaria, la cual no es deseada por el desarrollar. En la biblioteca la implementación de un puerto, no es más que escribir una clase derivada de `t3devlib::Port`, la cual sobrescribirá las funciones que sean necesarias.
- **Portabilidad:** Aunque las interfaces de TTCN-3 esta estandarizadas, existen algunas incompatibilidades entre los compiladores TTCN-3 de diferentes fabricantes. Las incompatibilidades pueden ocurrir por causa de errores o imprecisiones en el estándar, o simplemente porque algunos problemas de integración no están cubiertos por el mismo. En la biblioteca el usuario no tiene que preocuparse de las interfaces, ya que estas están implementadas en un módulo separado. Algunos compiladores son soportados y portar la biblioteca a otro compilador debería ser simple, ya que todos los cambios que deben hacerse se encuentran en el mismo lugar.
- **Manejo de excepciones y depuración:** Poder y precisión son características necesarias en un depurador para tener un corto ciclo de desarrollo y fiable. T3DevLib proporciona funciones de depuración para generar la traza de TRI/TCI en los intercambios con el Test Executable. También, los errores del CoDec están reportados con un conjunto de excepciones, que contienen un informe detallado de los errores y permite que sean fáciles de procesar.

2.2.4. Limitaciones

Todavía hay algunas limitaciones en el T3DevKit:

- La biblioteca no es compatible con los siguientes tipos:
  - float
  - universal charstring
  - address
  - object id
  - anytype
- Los subtipos primitivos de TTCN-3 (integers, octetstrings, etc.) no están soportadas por el CoDec Generator, deben ser declarados de forma manual en un archivo .h proporcionado por el usuario.
- Tipos y atributos de campos (excepto la palabra clave opcional) son ignorados por el CoDec Generator.
- La invocación `triSUTActionTemplate()` en TRI/SA no es soportada. También llamadas a procedimientos remotos de comunicación, como broadcast y multicast aún no están soportadas.

- Aunque esta listo para ser manejado internamente, los nombres de los módulos no están totalmente soportados. Nombres repetidos para identificadores de diferentes módulos deben evitarse al escribir Abstract Test Suites.
- Definiciones de tipos anidados no están soportadas.
- Las importaciones no son compatibles con el generador, todos los nombres de archivos fuentes TTCN-3 deben ser brindados manualmente.

2.3. TTworkbench (TTwb)

El Testing Technologies TTworkbench esta basado en tecnología Eclipse e incluye la posibilidad de instalar y utilizar sus extensiones.

TTworkbench tiene todas las características de un entorno integrado de desarrollo y ejecución (IDE) de tests, para cualquier tipo de proyecto de automatización [16][8].

2.3.1. Plataforma confiable para la automatización de tests

Esta herramienta puede ser utilizada para realizar pruebas de productos y servicios en una amplia gama de diferentes sectores de la industria.

El TTworkbench aparece en tres formatos de producto diferentes:

- TTworkbench Express: Ejecuta las especificaciones en los tests.
- TTworkbench Basic: Mantiene y modifica Test Suites, esta es versión utilizada en nuestro proyecto, gracias a la licencia académica otorgada.
- TTworkbench Professional: Construye entornos complejos de tests.

Beneficios principales:

- Concepto de desarrollo todo en uno.
- Rápida y fácil definición de tests en formato textual y/o gráfico.
- Soporte completo para la automatización de tests.
- Tecnología independiente de los sistemas de test diseñados en TTCN-3.
- Incluye chequeos en etapas tempranas del diseño de tests.
- Alta re-utilización y fácil ejecución de los tests predefinidos.
- Rápida integración de los sistemas de test.
- Fiable en el manejo de la traza de los Test Suites, rápido y eficaz en seguimiento de errores.
- Extensible a partir de una gran variedad de plugins.

2.3.2. Características destacadas del TTworkbench

A continuación, se enumeran las funcionalidades más importantes brindadas por el entorno de desarrollo.

- **TTman**

  Esta perspectiva dentro del TTwb, permite la gestión, ejecución y análisis de los test suites TTCN-3 compilados, sus aplicaciones principales son:

  - Definición, ejecución, almacenamiento y reanudación de campañas de test en "proyectos inteligentes".
  - Logging textual y gráfico con diferentes niveles.
  - Vista de los datos definidos en los tests y soporte para su análisis.
  - Funciones de seguimiento de fuentes TTCN-3.
  - Selección y configuración del Test Adapter y los agregados utilizados en tiempo de ejecución (puertos, CoDecs, funciones externas)
  - Vista para el despliegue de estadísticas.
  - Generación de reportes sobre los tests (HTML, PDF, Excel, Word).
  - Secuencias de comandos para el modo de pruebas por lotes.
  - Opción para el uso de linea de comandos.

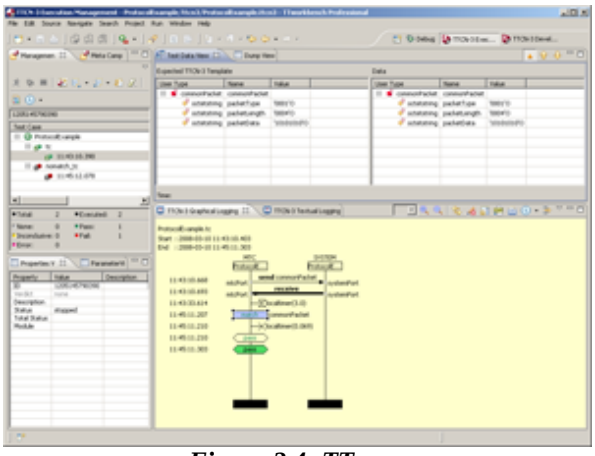


Figura 2.4: TTman

- **CL Editor**

  La perspectiva TTCN-3 Development incluye este editor, basado en las definiciones de tests (incluyendo T3Doc) y proporciona las siguientes funcionalidades:

  - Soporte completo de los estándares TTCN-3 ETSI.
  - Formateo del texto y resaltados de la sintaxis del lenguaje.
  - Anotaciones sobre el texto y reporte de errores al momento de recorrer el código.

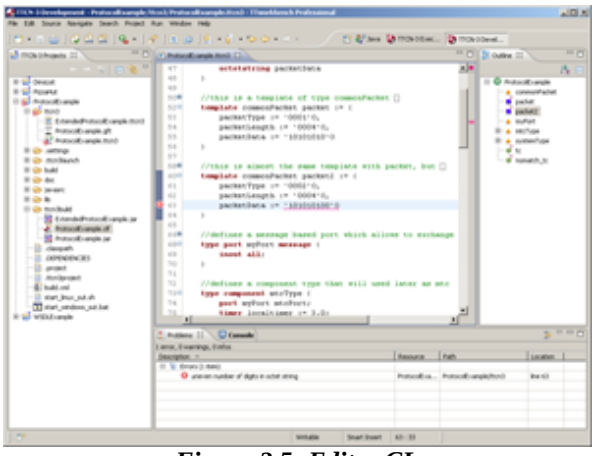


Figura 2.5: Editor CL

- Manejo de código plegable y plantillas de código.
- Generación de datos TTCN-3 y asistente para su modificación.
- Asistencia/sugerencia para el contenido del código.
- Vista para el despliegue de las especificaciones de estructuras (Outline).
- Soporte para solución rápida de errores.
- Refactorización.
- Exportación a T3Doc en HTML.

- **TTthree**

  Es un conjunto de componentes Java que permiten la compilación de los módulos TTCN-3 dentro de test ejecutables, brindan las siguientes funcionalidades:

  - Compilación y ejecución independiente de la plataforma.
  - Soporte completo de los estándares TTCN-3 ETSI.
  - Adaptación flexible para los tests de dispositivos vía la estandarizada TTCN-3 Runtime Interface (TRI).
  - Rápida integración de CoDecs externos vía la estandarizada TTCN-3 Control Interfaces (TCI).
  - Opción para el uso de línea de comandos.

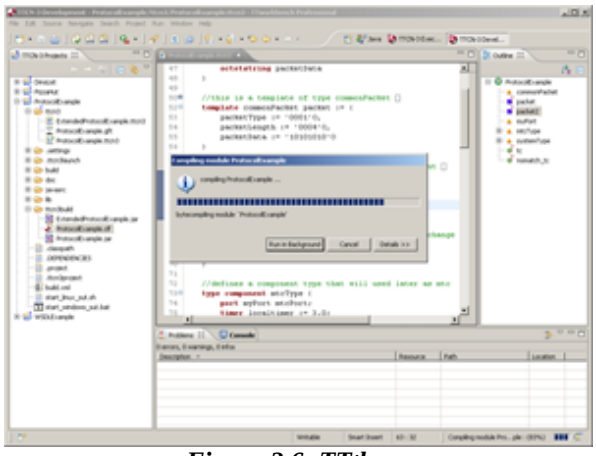


Figura 2.6: TTthree

- **FT Editor**

  Este es un sencillo editor en modo gráfico para la especificación de tests y su documentación, las características principales son:

  - Diseño gráfico y visualización de los casos de test como secuencia de diagramas GFT.
  - Importación de los datos nativos TTCN-3.
  - Generación en linea del TTCN-3 Core Language.
  - Generación automática de GFT (gráficos) fuera del TTCN-3 Core Language.

- Exportación flexible de gráficos a imágenes GIF (documentación).

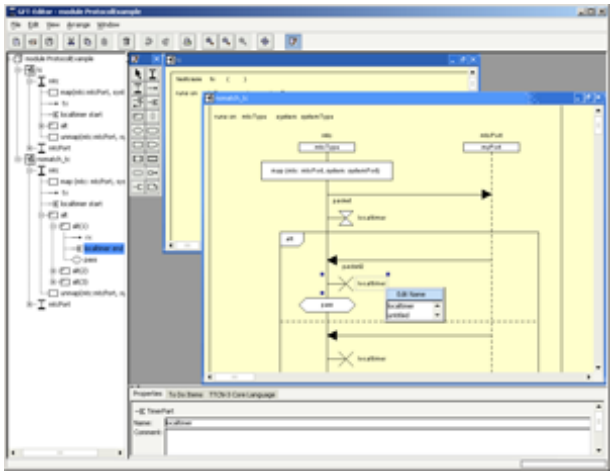


Figura 2.7: Editor GFT

- **TTdebug**

  Esta perspectiva dentro del TTwb, permite contar con un depurador para el código fuente TTCN-3, dentro de sus funcionalidades se encuentran:

  - Depuración transparente de TTCN-3, Test Adapters y CoDecs basados en Java simultáneamente.
  - GUI basada en agregado/quitado de breakpoints y watchpoints.
  - Suspensión/Reanudación manual de los test suites que se están corriendo.
  - Paso a paso en un Test suite suspendido.
  - Vista del estado y stack traces de múltiples componentes.
  - Visualización de contenido y modificación de las variables y los componentes locales desde la base o estructura de tipos y de parámetros locales

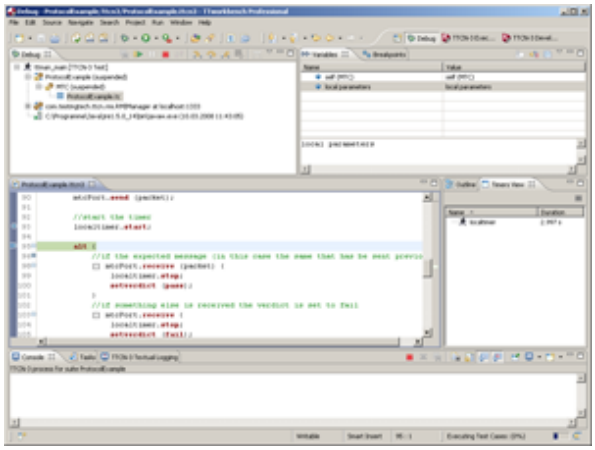


Figura 2.8: TTdebug

  - Vista del estado de los timers y activación manual de timeouts.
  - Vista del estado de la colas en los puertos.
  - Manipulado del orden de los mensajes en las colas.
  - Visualización del contenido de los mensajes en las colas.

- **RPDE - Runtime Plugin Development Environment**

  El módulo Runtime Plugin Development Environment posibilita el desarrollo de extensiones, su características principales son:

  - Intuitivo marco de trabajo para el desarrollo TTCN-3 plugins de extensiones de tiempo de ejecución.
  - Edición en un modo plano, así como en un modo de GUI asistida para todas las funcionalidades de un TTCN-3 plugin de tiempo de ejecución.

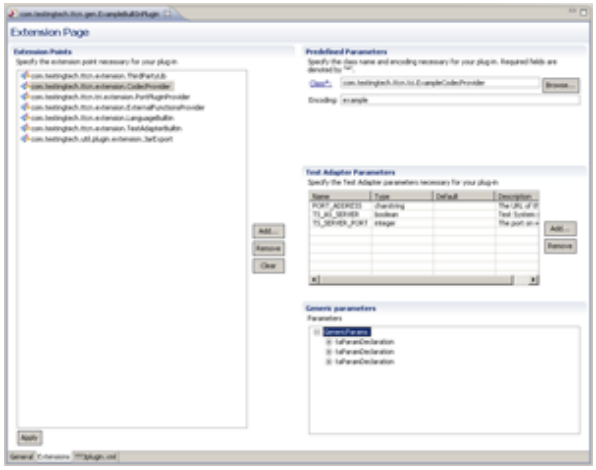


Figura 2.9: RPDE

3. Descripción del Problema

En esta sección se realiza la introducción a la problemática propuesta en este proyecto, intentando dar la pautas necesarias para la comprensión de las tecnologías contempladas, evaluadas y seleccionadas. También se brindan un compendio de los requerimientos involucrados en el alcance y algunos detalles sobre como interactúan los módulos integrados.

Dado que, se cuenta con el compilador Java TT (TestingTech) y la herramienta para TTCN-3 CoDec Generator C/C++ de IRISA, se muestra como evolucionaron las ideas iniciales hasta llegar a pensar en el conector JNI. Todo esto motivó la generación de una pieza de software que permita la re-utilización de componentes existentes, interoperando entre las plataformas Java y C++.

Se describen los requerimientos, donde se mencionan términos como: alto grado de mantenibilidad de los productos, simplicidad en las entidades y módulos que componen del conector.

Otro punto a ser atacado es la descripción de las interfaces estandarizas, y la independencia entre las herramientas en cada una de las plataformas, logrando así, ser mantenidas en forma separada.

3.1. Planteo inicial

Al comienzo, se conocía la existencia de una herramienta para la generación automática de CoDecs (T3DevKit con su CDGen, descrita en el apartado 2.2.), la cual cuenta con un parser de TTCN-3 y en su evolución intenta adaptarse a la funcionalidades implementadas por Go4IT.

En base al problema que resuelve el T3DevKit, es que se comenzó a trabajar, estudiando las diferentes soluciones a nivel de diseño e implementación para cubrir las mismas prestaciones de esta herramienta, pero dentro de la plataforma Java.

Como ideas iniciales surgen varias posibilidades: tomar como base las estructuras y funciones implementadas en C++ y a partir de las mismas re-implementar sus símiles en Java, generar una pieza de software que tradujera el código C++ a Java re-utilizando las funcionalidades existentes y como otra opción, tomando en cuenta que se contaba con una licencia para el uso del compilador incluido en TTworkbench de Testingtech, se pensó en analizar la implementación del mismo y tomar como base este compilador

para cubrir el alcance necesario al momento de contar con un compilador Java de código abierto.

Después de analizar todos los puntos mencionados, se trazó como objetivo inicial poder comprender los tipos TTCN-3 definidos por los estándares y como se realizan las respectivas correspondencias con los lenguajes de plataforma, tanto para C++ como para Java.

Luego de esto, los objetivos del proyecto fueron cambiando, ante la posibilidad de integrar e interoperar entre los componentes de software existentes y se planteó como meta importante el re-uso de módulos en C++ funcionales al momento, lograr código que sea mantenible y la posibilidad de contar con una plataforma dual, que cubra lo necesario para el manejo del lenguaje TTCN-3. Esto implicaba la implementación de otro módulo, un componente que sea capaz de implementar las interfaces definidas para los dos lenguajes, realice las conversiones entre las definiciones de tipos C/C++ a clases Java y viceversa, permita las invocaciones desde un lenguaje al otro con transparencia reutilizando así las funcionalidades implementadas.

Lo descrito, dio lugar a que el nuevo alcance para el proyecto fuera, la implementación y adecuada documentación de un conector utilizando la tecnología JNI, que oficiará de "puente" entre ambos lenguajes de base.

3.2. Requerimientos sobre el conector

En este trabajo se deben realizar las adaptaciones e interfaces de software necesarias para permitir la utilización del T3DevKit en un entorno TTCN-3 Java.

Como ya se estableció anteriormente el T3DevKit posee dos componentes principales: CoDec Generator (CDGen) y T3DevLib. Se busca que la implementación generada por el CDGen en C sea utilizable en el entorno Java. La implementación de la TCI/CD generada por el CDGen en principio requiere que se realice solamente un wrapper para las funciones TCI/CD, pero si se quieren utilizar todas las características de la T3DevLib debemos proveer además wrappers para las funciones TRI. La T3DevLib también permite el acceso a funciones externas (TRI/PA), así como, provee la implementación de puertos a nivel Ethernet (TRI/SA), utilizado en los test cases IPv6 que acompañan al T3DevKit.

En concreto el trabajo consiste en re-usar el CoDec Generator en Java como una forma inicial de aprovechar funcionalidades existentes, realizando un enmascaramiento JNI del CDGen; para lograr esta funcionalidad se deben "traducir" todas las llamadas TT → CDGen y CDGen → TT y se debe cumplir con la implementación de la interfaces definidas por Testingtech, algunas son de ETSI (las estandarizadas) y otras de TT, para generar un componente genérico y que no se tenga que definir por parte de cada usuario. Este es el principal objetivo de diseño, que tal vez, no se lleve a cabo por completo por dependencias de TT.

Otro de los objetivos es la evaluación de la posibilidad de que el compilador invoque a TRI/PA, TRI/SA y TCI/CD, de manera nativa y transparente, dado que la utilización de las interfaces TCI y TRI, debería facilitar la integración con el mismo, ya que las APIs utilizadas son las mismas.

También se busca, fomentar la re-utilización de código funcional y de esta manera acotar el esfuerzo de implementación, ya que mantener una única pieza de software reduce el costo de mantenimiento, evitando replicar los cambios y ajustes al T3DevKit para dos versiones del mismo en lenguajes diferentes.

3.3. Integración con TTCN-3

La Figura 3.1 muestra un esquema introductorio de cómo se integra una pieza de software que cumple nuestros requerimientos con las entidades principales en tiempo de ejecución de TTCN-3.

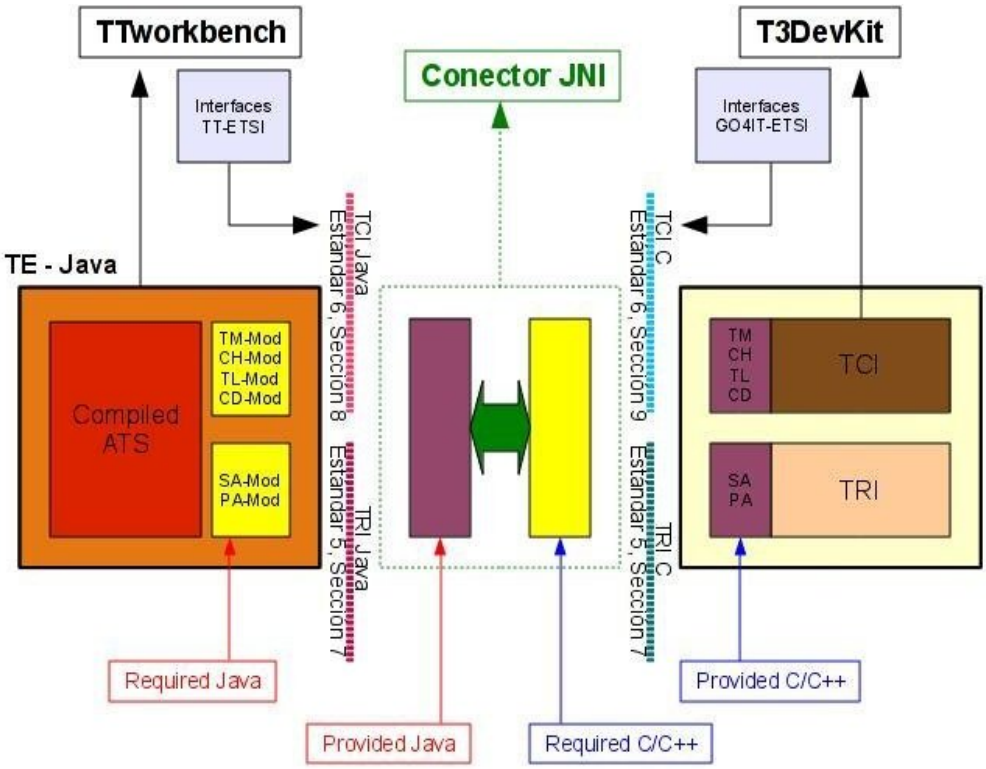


Figura 3.1: Diagrama de interacción entre módulos con enfoque en el Conector

El diagrama está centrado en un conector y como éste interactúa con las entidades definidas genéricamente en TTCN-3, en nuestro caso, implementadas por TTwb y T3DevKit. Los conceptos presentados en las figuras 2.1, 2.2 y 2.3, se mantienen. La duplicación aparente se debe a que deberán existir implementaciones Java y C/C++ de todos los componentes.

Se comienza desde el TE generado por el compilador, que necesita utilizar las interfaces Provided Java (TT/ETSI), esto implica "atravesar" el conector que de forma nativa, invoca a las interfaces Provided en C (Go4IT-ETSI) para interactuar con la T3DevLib, que como se muestra en la Figura 2.3, brinda la implementación de TCI/CD, TRI/SA y TRI/PA en C/C++; pero también, para interactuar con el TE necesita utilizar las interfaces Required C (Go4IT-ETSI), lo que lleva a "atravesar" el conector en

sentido inverso, llamando a las interfaces Required en Java (TT-ETSI) provistas por el TTwb.

Se debe notar que, en la Figura 3.1 se incluyen las interfaces TCI/TM, TCI/CH y TCI/TL, pero éstas no son soportadas por el T3DevKit y no fueron incluidas en el alcance de este proyecto; también, fueron mencionadas las secciones y partes de los estándares donde se encuentran las correspondencias de la interfaces para Java, o lo que es lo mismo, las operaciones que aparecen en ambos lados del conector.

Para el mejor entendimiento de como el conector se adecua a la arquitectura TTCN-3, se dispone del diagrama en la Figura 3.2.

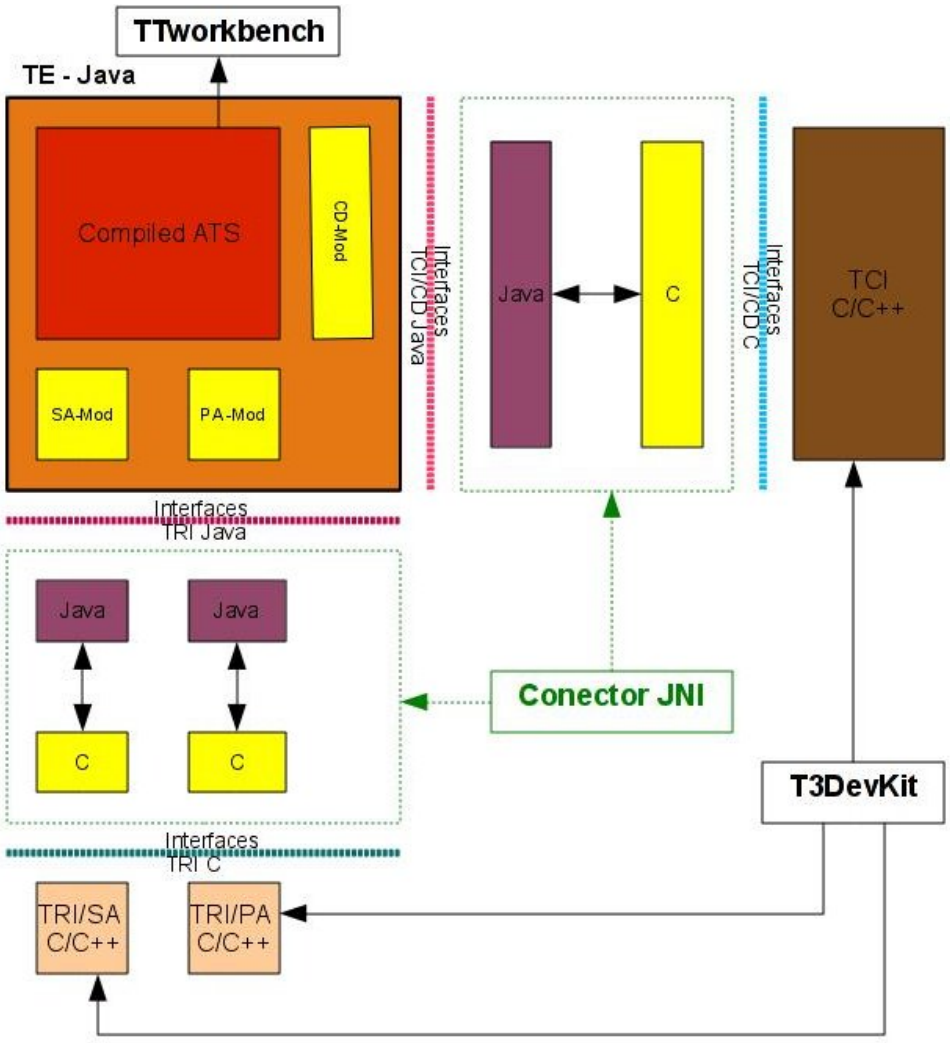


Figura 3.2: Diagrama de interacción entre módulos con enfoque en la arquitectura TTCN-3

Como lo indica la Figura 2.1, en la arquitectura TTCN-3 el TE está rodeado por las entidades participantes en la ejecución; entonces, en el diagrama de la Figura 3.2 se muestra como el TE Java interactúa con las entidades CD, PA y SA por medio de interfaces Java, que luego, el conector se encarga de corresponder con interfaces C en

forma bidireccional; o sea, realiza las correspondencias para las operaciones Provided (Java → C/C++) y Required (C/C++ → Java).

3.4. Detalles específicos de TTCN-3 y el T3DevKit

A lo largo de la definición del lenguaje TTCN-3, se determinan las interfaces con los componentes externos de forma abstracta, aparecen detalladas las características de las operaciones y sus respectivas correspondencias a los lenguajes de plataforma.

Todo el tiempo se trato de seguir la evolución de los estándares, si bien en este trabajo se garantiza la compatibilidad con la versión 3.2.1, incluida en las referencias de este documento.

TTCN-3 Control Interfaces (TCI) [12], es el conjunto de interfaces que definen la interacción del TE con el manejador del test, la codificación y de-codificación y el manejo de componentes en el sistema de testeo. Estas interfaces proveen una adaptación estandarizada de un sistema bajo testeo para una plataforma particular. Las definiciones de las interfaces son completamente independientes del lenguaje de plataforma .

La entidad CD es responsable de la codificación y de-codificación externa de valores TTCN-3 en las cadenas de bits adecuadas para ser enviadas al SUT. Cuando se usan CoDecs externos, el TE determina que CoDecs se utilizarán. Pasa los datos TTCN-3 al codificador adecuado para obtener los datos codificados. Los datos recibidos son decodificados en la entidad CD por intermedio del decodificador adecuado, que traduce los datos recibidos a valores TTCN-3.

Dentro de la especificación de TCI, se encuentra la TCI Data, como un conjunto de tipos de datos abstractos. Estos describen, en alto nivel, que tipo de datos se transmiten en una llamada a una entidad. Los tipos de datos abstractos son utilizados para determinar como los datos TTCN-3 se transmiten desde el TE hacia un codificador, el cual deberá convertirlo de valor TTCN-3 a su representación de cadena de bits; y el caso inverso; como los datos transmitidos desde un decodificador hacia el TE serán convertidos a partir de su representación en cadena de bits a un valor TTCN-3 ([12] en su sección 7.2).

La representación concreta de estos tipos abstractos de datos, así como la definición de tipos de datos básicos, se definen en las respectivas correspondencias con los lenguajes de plataforma en las secciones 8 y 9 de la referencia [12].

A modo de ejemplo del nivel de abstracción manejado por el lenguaje TTCN-3, presentaremos la definición del tipo de datos entero del lenguaje. En la sección 6.1.0 de la especificación del lenguaje TTCN-3 [14] encontramos la siguiente definición:

> *a) integer: a type with distinguished values which are the positive and negative whole numbers, including zero.*
> *Values of integer type shall be denoted by one or more digits; the first digit shall not be zero unless the value is 0; the value zero shall be represented by a single zero.*

Como podemos observar, la definición abstracta de un entero no está acotada, como usualmente lo está el entero en los lenguajes de programación estándar.

Complementando lo ejemplificado, se muestran las correspondencias para el tipo entero TTCN-3 en Java y C, con los respectivos métodos de acceso a la representación .

En Java, el tipo Tinteger IDL se corresponde con el tipo básico `int` y la siguiente interfaz resuelve la correspondencia para el tipo `IntegerValue` TTCN-3:

```
package org.etsi.ttcn.tci;
public interface IntegerValue {
        public void setInteger(int value);
        public int getInteger();
}
```

Los métodos son descritos a continuación:

- `setInteger(int value)` - Carga el `IntegerValue` con el valor int del parámetro '`value`'.
- `getInteger()` - Retorna el valor int representado por este `IntegerValue`.

En tanto, en ANSI C se utiliza la interfaz TCI IDL, donde los métodos `TInteger getInt()` y `void setInt(in TInteger value)` llevan a cabo la correspondencia; aquí el `IntegerValue` TTCN-3 esta representado por el tipo `TciValue` genérico, sobre el cual se aplican un conjunto de operaciones que especifican la representación de un entero.

Las operaciones correspondientes a `getInt()` son:

- `String tciGetIntAbs(TciValue inst)` - Retorna el valor absoluto del entero (en base 10) como una cadena de caracteres ASCII.
- `unsigned long int tciGetIntNumberOfDigits(TciValue inst)` - Retorna el número de dígitos en el valor entero.
- `Boolean tciGetIntSign(TciValue inst)` - Retorna `true` si el número es positivo, en cualquier otro caso `false`.
- `char tciGetIntDigit (TciValue inst, unsigned long int position)` - Retorna el valor del dígito en la posición determinada por el parámetro '`position`', donde la posición 0 corresponde al dígito menos significativo.

Las operaciones correspondientes `setInt()` son:

- `void tciSetIntAbs(TciValue inst, String value)` - Carga el valor absoluto del entero (en base 10) a partir de una cadena de caracteres ASCII. El primer dígito debe ser 0 solamente si el valor es 0.
- `void tciSetIntNumberOfDigits(TciValue inst, unsigned long int dig_num)` - Carga el número de dígitos del valor entero.
- `void tciSetIntSign(TciValue inst, Boolean sign)` - Carga el signo en + (`true`) o - (`false`).

- `void tciSetIntDigit(TciValue inst, unsigned long int position, char digit)` - Carga el valor del dígito en la posición determinada por el parámetro `'position'`, donde la posición 0 corresponde al dígito menos significativo.

La representación abstracta de un entero no esta acotada, en tanto que representaciones concretas en Java y C si lo están, por lo que hay perdida de precisión al realizar las correspondencias. La manera de representar un entero en cada lenguaje es totalmente diferente y no es trivial el pasaje de una a la otra.

Se debe notar que los valores de cualquier tipo de datos cuentan con un identificador único dentro de la implementación del Test System, donde la unicidad se mantiene de forma global y en cualquier momento de la ejecución. Esto garantiza que los distintos objetos, por ejemplo, dos temporizadores, sean localizados por diferentes identificadores y además los identificadores no sean re-utilizados.

Para estos tipos de datos abstractos se definen un conjunto de operaciones para realizar el proceso de codificación/de-codificación. Funcionalmente cada tipo de datos se define por un conjunto de operaciones que lo acompañan. Las operaciones sobre tipos abstractos retornan, ya sea un valor de este tipo abstracto o un tipo básico como por ejemplo: `boolean`.

Todas las operaciones han sido definidas utilizando Interface Description Language (IDL). La correspondencia concreta de las operaciones en los lenguajes de plataforma esta dada en las secciones 8 y 9 de la referencia [12]. En ciertos lenguajes, la aplicación de una operación sobre un tipos de dato abstracto se representa por el pasaje (por valor o por referencia, dependiendo de la correspondencia) del valor concreto como parámetro de la operación, otros lenguajes, pueden elegir otras formas de referenciar un valor concreto. Para indicar la imposibilidad de realizar una determinada tarea o para indicar la ausencia de un parámetro opcional, el valor particular 'null' es utilizado. Este se puede considerar como un valor especial reservado. El lenguaje de plataforma seleccionado definirá la representación concreta del valor 'null'.

La representación abstracta de los tipos y valores TTCN-3 cuenta con dos partes:

- Un tipo de dato abstracto `Type`, el cual representa todos los tipos TTCN-3 en un módulo TTCN-3.
- Diferentes tipos de datos abstractos que representan los valores TTCN-3, por ejemplo: valores TTCN-3 esta dados por tipos TTCN-3. Estos pueden ser tipos predefinidos de valores TTCN-3 o de tipos TTCN-3 definidos por el usuario.

Para el acceso, evaluación y codificación de datos TTCN-3, el Test System utiliza el tipo de dato abstracto `Type` y los distintos valores abstractos de los tipos. De esta forma, los tipos de datos abstractos indican el nivel de abstracción entre el TTCN-3 Executable (TE) y el resto del Test System utilizando las interfaces TCI.

El la Figura 3.3 se muestra el diagrama de clases con la jerarquía de valores abstractos.

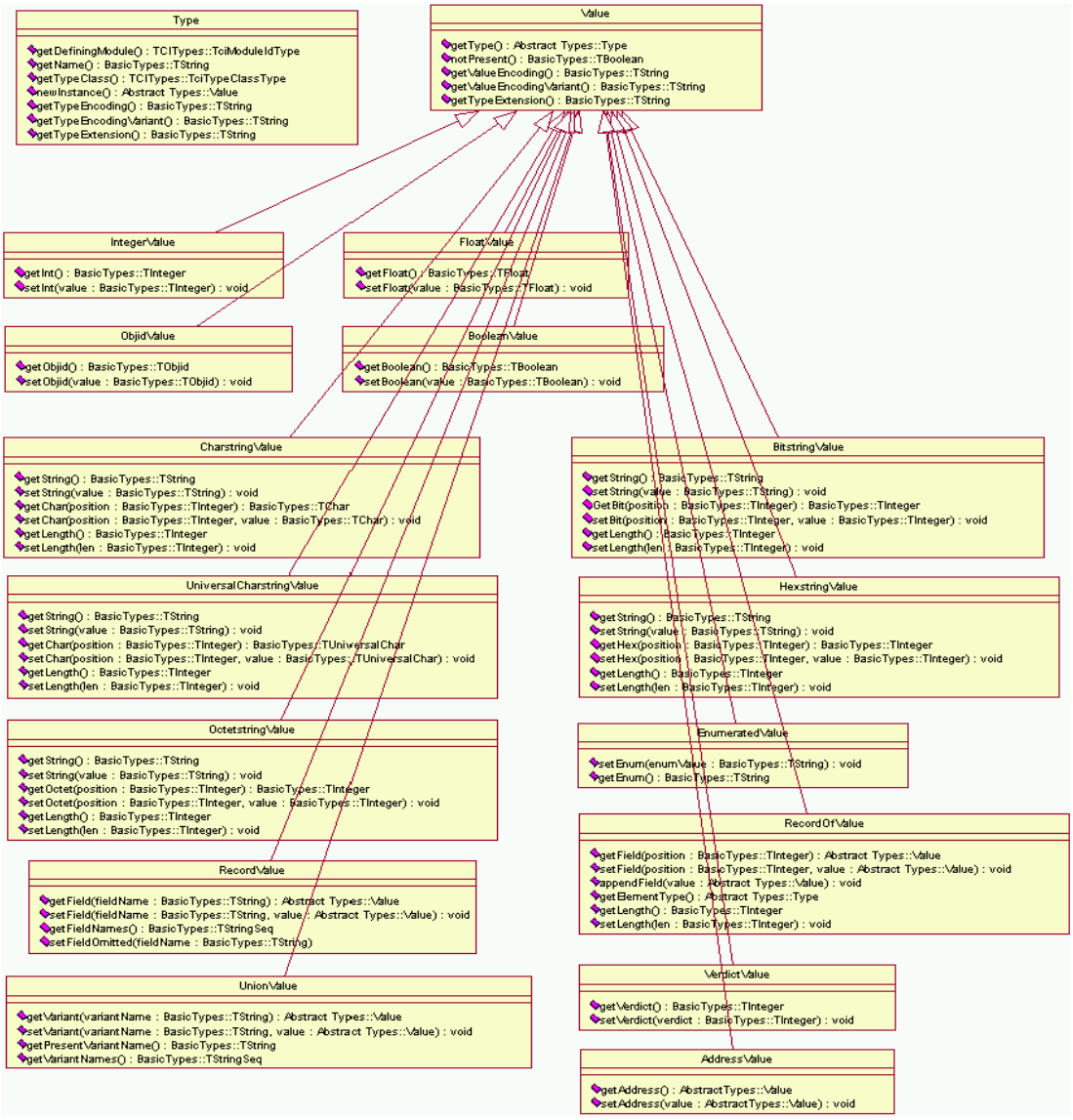


Figura 3.3: Jerarquía de TCI Values en su representación abstracta.

TTCN-3 Runtime Interface (TRI) [13], es el conjunto de interfaces que definen la interacción del TTCN-3 Executable con el SUT y los adaptadores de la plataforma en el sistema de testeo. Esta interfaces proveen una adaptación estandarizada para temporizadores y comunicación de un sistema de testeo hacia el procesamiento particular de la plataforma y el SUT respectivamente; las definiciones de las mismas son completamente independientes del lenguaje de plataforma.

El System Adapter (SA), adapta los mensajes y procedimientos basados en la comunicación del sistema de test TTCN-3 con el SUT para una plataforma de ejecución particular. Realiza la correspondencia entre los puertos y componentes de comunicación del Test Case TTCN-3 con las interfaces de los puertos en el sistema bajo pruebas, e implementa las interfaces reales del sistema de test. Es responsable de propagar envíos,

respuestas y operaciones desde el SUT hacia el TE y desde el TE hacia el SUT, notificando al TE de cualquier evento del test recibido en las colas mantenidas por sus puertos.

Los procedimientos basados en operaciones de comunicación con el SUT están definidos en el SA, el cual es responsable de distinguir entre los diferentes mensajes dentro de los procedimientos de comunicación (por ejemplo: call, reply y exception) y de propagar los mismos de manera adecuada hacia el SUT o el TE. La semántica de los procedimientos TTCN-3 basados en la comunicación, por ejemplo: el efecto de una operación sobre un componente de test TTCN-3 en ejecución, son manejados por el TE.

El SA mantiene una interfaz con el TE, que se utiliza para enviar mensajes del SUT hacia la SA y para el intercambio de datos codificados de prueba entre las dos entidades en las operaciones de comunicación con el SUT.

El Platform Adaptor (PA) ejecuta funciones externas TTCN-3 y proporciona una noción simple del tiempo para un sistema de test TTCN-3. En esta entidad, las funciones externas son implementadas y también todos los temporizadores. Las instancias de temporizadores son creadas en el TE. Un temporizador en el PA sólo puede ser distinguido por su Timer IDentification (TID), por lo tanto, se trata a los temporizadores explícitos e implícitos de la misma manera.

La interfaz con el TE permite la invocación de funciones externas y el inicio, lectura, y la parada de temporizadores, así como también, permite saber el estado de los mismos a partir del ID de temporizador. La PA informa de la espiración de temporizadores al TE.

Partiendo de esta base teórica, donde se cuenta con una misma especificación abstracta, se desprende la posibilidad de mantener un conector para la implementación de la comunicación entre módulos escritos en ambos lenguajes.

El T3DevKit implementa las interfaces Go4IT (ver apartado 2.2.2), las cuales realizan la correspondencia con las estructuras y funciones en C/C++, mientras que el desarrollo de este proyecto lleva a cabo la correspondencia con las estructuras y funciones en Java.

Para la compilación e instalación de la T3DevLib, se utiliza la herramienta Automake [17], por lo que fue necesario adquirir conocimientos sobre la mismas para poder realizar la adaptaciones correspondientes a las dependencias con respecto al compilador TTCN-3.

3.5. Resumen

Debido a que hay implícita una correspondencia funcional y de tipos de datos entre las APIs C y Java es posible implementar una pieza de software (la cual denominamos indistintamente wrapper o conector) que traduce las llamadas de una API a la otra, pudiendo así re-utilizar el CoDec Generator en Java, realizando un enmascaramiento JNI del T3DevKit y traduciendo todas las llamadas entre las dos plataformas.

Se mantiene la posibilidad de que el compilador invoque a TRI-PA, TRI-SA y TCI-CD, de manera nativa y transparente, dado que la utilización de las interfaces estandarizadas TCI y TRI, facilitan la integración con el mismo.

El principal objetivo es crear un componente genérico que no tenga que ser definido por cada usuario; que tal vez, no implemente todas las correspondencias con las funcionalidades provistas por el T3DevKit, a consecuencia de las dependencias introducidas por el compilador TT. Este componente fomenta la re-utilización de código haciendo más fácil el mantenimiento y minimizando el esfuerzo de desarrollo.

4. Diseño del conector

En esta sección se muestra como se ha diseñado el conector, como se ha llegado al mismo, evaluando y mostrando el uso de la tecnología JNI. Se mencionan características de los módulos C/C++ y Java, se muestran las interfaces utilizadas para la comunicación entre las distintas plataformas.

Se muestran las responsabilidades de cada módulo, así como también, con que funcionalidades utilitarias se cuenta.

4.1. Descripción

Para el logro de los objetivos trazados, se planteó la opción de tener una implementación nativa Java y conexión JNI para el manejo de las invocaciones entre módulos.

Al invocar al CDGen, este necesita obtener información desde la parte TTCN-3, por lo cual se deben hacer invocaciones a través de la interfaz TCI/CD en la operación de "encode" para obtener los valores de los campos, etc.; y en "decode" para generar el tipo TTCN-3.

Además es necesario contar con la funciones de comunicación y manejo de temporizadores implementadas en la T3DevLib, por lo cual se deben hacer invocaciones a través de las interfaces TRI/SA (en operaciones básicas como: `triSAReset()`, `triExecuteTestcase()`, `triSend()`, `triMap()`, `triUnmap()` o `triEndTestCase()`) y TRI/PA (en operaciones básicas como: `triPAReset()`, `triStartTimer()` o `triStopTimer()`).

Todas las llamadas que se realizan desde CDGen son sobre los objetos que están "del otro lado" de su API, es decir, del correspondiente Test System. La responsabilidad de los módulos C es convertir las estructuras de datos a objetos JNI, para luego, invocar a las interfaces Java con estos objetos como parámetros.

La Figura 4.1 muestra el diagrama de clases general (luego se mostrarán las entidades participantes con más detalles), donde `TciCDProvided`, `AbstractBaseCodec`, `TriCommunicationSA`, `TriCommunicationPA`, `TestAdapter`, `TriCommunicationTE`, `TriPlatformTE` y `TciCDRequired` representa las clases ETSI/TTwb; mientras que `CoDec`, `AbstractBaseAdapter` y `Adapter` son las clases necesarias para el conector.

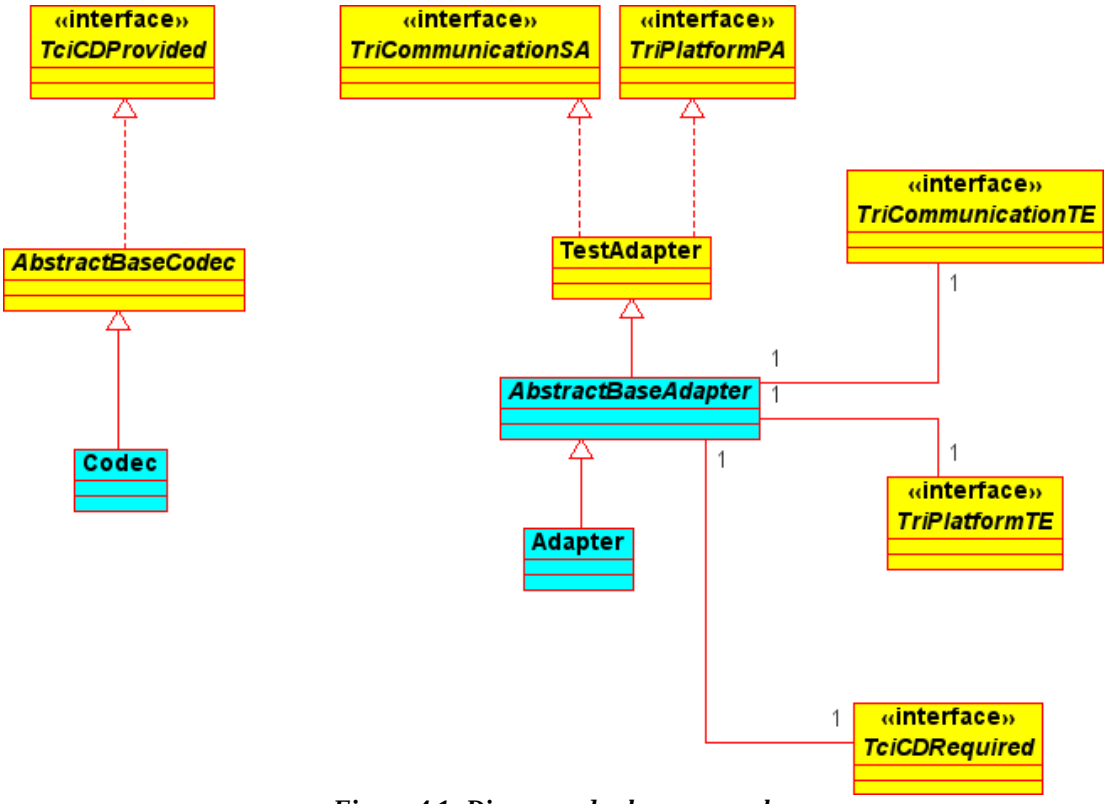


Figura 4.1: Diagrama de clases general

4.2. Diseño del Codec

La clase `com.testingtech.ttcn.tci.codec.base.AbstractBaseCodec` provee la implementación base para la codificación y de-codificación de valores TCI. Esta clase indica como se realiza la codificación y de-codificación. Al extenderla se pueden utilizar las funcionalidades implementadas y/o sobrescribir las mismas para obtener un CoDec apropiado, necesitando un conocimiento específico de los tipos estructurados que se utilizarán en el Test Suite.

Esta clase es provista por TT (TestingTech) e implementa la interfaz TCI/CD Provided de ETSI, de la misma parte el diseño de las clases incluidas en este trabajo.

La Figura 4.2 muestra el diagrama de clases participantes en los procesos de codificación y de-codificación de tipos TTCN-3, las clases `TciCDProvided` y `AbstractBaseCodec` pertenecen al TT, mientras que la clase `Codec` es incluida para llevar a cabo la interacción entre los módulos en forma dual.

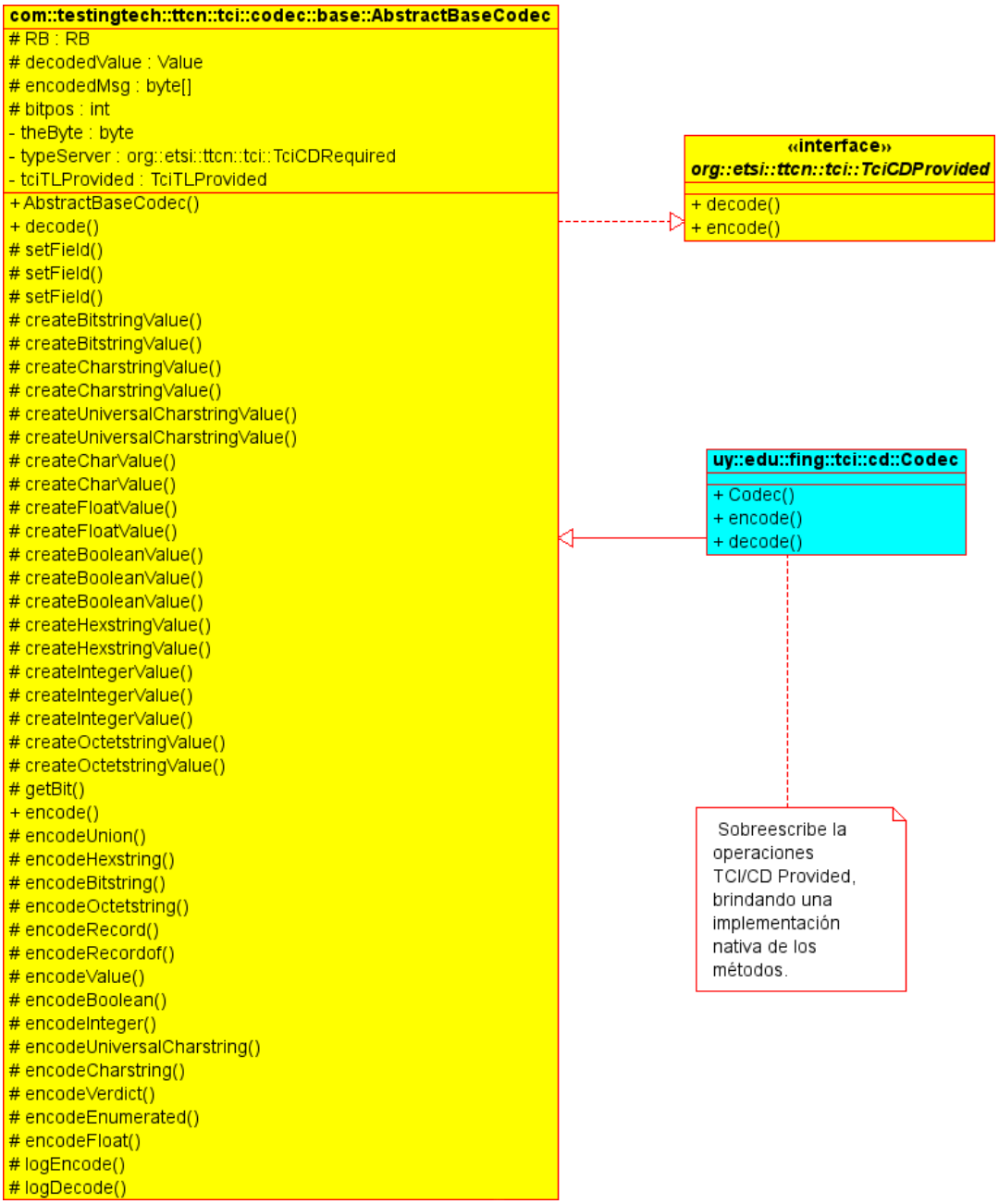


Figura 4.2: Diagrama de clases para el Codec.

La especificación detallada de los métodos (valores de retorno, argumentos, etc.) esta disponible en la referencia [12].

La importancia de los valores TCI radica en que estos determinan como los datos incluidos en un test TTCN-3 se pasan desde el TE hacia el CoDec y viceversa, lo que nos lleva a atacar el estudio de las estructuras que los representan. De esta manera para realizar un diseño que cumpla con los objetivos trazados, se estudió de manera

pormenorizada el conjunto de clases utilizadas por TT y por el lado del T3DevKit el conjunto de estructuras de datos disponibles.

Tomando en cuenta lo descrito antes, con respecto a la clase base utilizada por el TTwb y la importancia de convertir los valores TCI, se agrega la clase Codec, la cual participara en la interconexión entre módulos implementados en distintos lenguajes.

La clase concreta `uy.edu.fing.tci.cd.Codec` extenderá a la clase abstracta `com.testingtech.ttcn.tci.codec.base.AbstractBaseCodec` como se muestra en la Figura 4.2. De esta manera se heredan las funcionalidades brindadas por TTwb y permite que se puedan sobrescribir los métodos de la interfaz TCI/CD Provided de manera nativa, logrando que de manera transparente se invoquen de Java los métodos implementados por el T3DEvKit en C/C++ .

Posicionándonos ya dentro del conector JNI, se tienen un conjunto de operaciones correspondientes a la clase Codec Java con métodos nativos. Esto da lugar al diseño de un módulo C que implementa las mismas.

La responsabilidades del módulo Codec en lenguaje C son: incluir el manejo de las bibliotecas JNI y a partir de las estructuras JNI recibidas, realizará la traducción a las estructuras definidas en la interfaces del T3DevKit, para luego invocar las funcionalidades brindas por la T3DevLib; luego se tiene el caso inverso, donde se toman las estructuras retornadas por el T3Devkit y se convierten a objetos JNI, los cuales sirven como valores de retorno a la plataforma Java.

4.3. Diseño del Adapter

La clase `com.testingtech.ttcn.tri.TestAdapter` es una implementación básica de un adaptador al Test Suite. Cada adaptador que extienda de esta clase, contará con las funcionalidades básicas del TRI, como lo es la intercomunicación entre los componentes del test.

Esta clase es provista por TestingTech e implementa la interfaces TRI/PA Provided y TRI/SA Provided de ETSI, cabe mencionar que se encarga de implementar las interfaces para le manejo del debugger a nivel de PA (correspondiente a la arquitectura de TT, en este trabajo no es abarcado este tema, sólo se reutilizan estas funcionalidades) y además, implementa una última interfaz `com.testingtech.ttcn.tci.TciEncoding` (correspondiente a la arquitectura de TT) que permite parametrizar la entidad que representara al CoDec básico, por lo que este comportamiento es fundamental a la hora de la integración con TTwb, y por lo tanto, es contemplado a la hora de diseñar el Adapter incluido en este trabajo.

Un Test System de TTCN-3 puede ser pensado conceptualmente como un conjunto de entidades que interactúan, donde cada entidad corresponde a un aspecto particular de las funcionalidades de un sistema de prueba en su implementación. Estas entidades gestionan la ejecución del test, ejecutan o interpretan código TTCN-3 compilado, realizando una correcta comunicación con el SUT, implementan funciones externas y manejan los temporizadores.

La Figura 4.3 muestra el diagrama de clases participantes en los procesos de comunicación y temporización en TTCN-3, donde `TriCommunicationSA`,

`TriCommunicationPA`, `TestAdapter`, `TriCommunicationTE`, `TriPlatformTE` y `TciCDRequired` son las clases del TT y las clases `AbstractBaseAdapter`, `Adapter` y `ConfigAdapter` son incluidas para la interacción entre los módulos en forma dual.

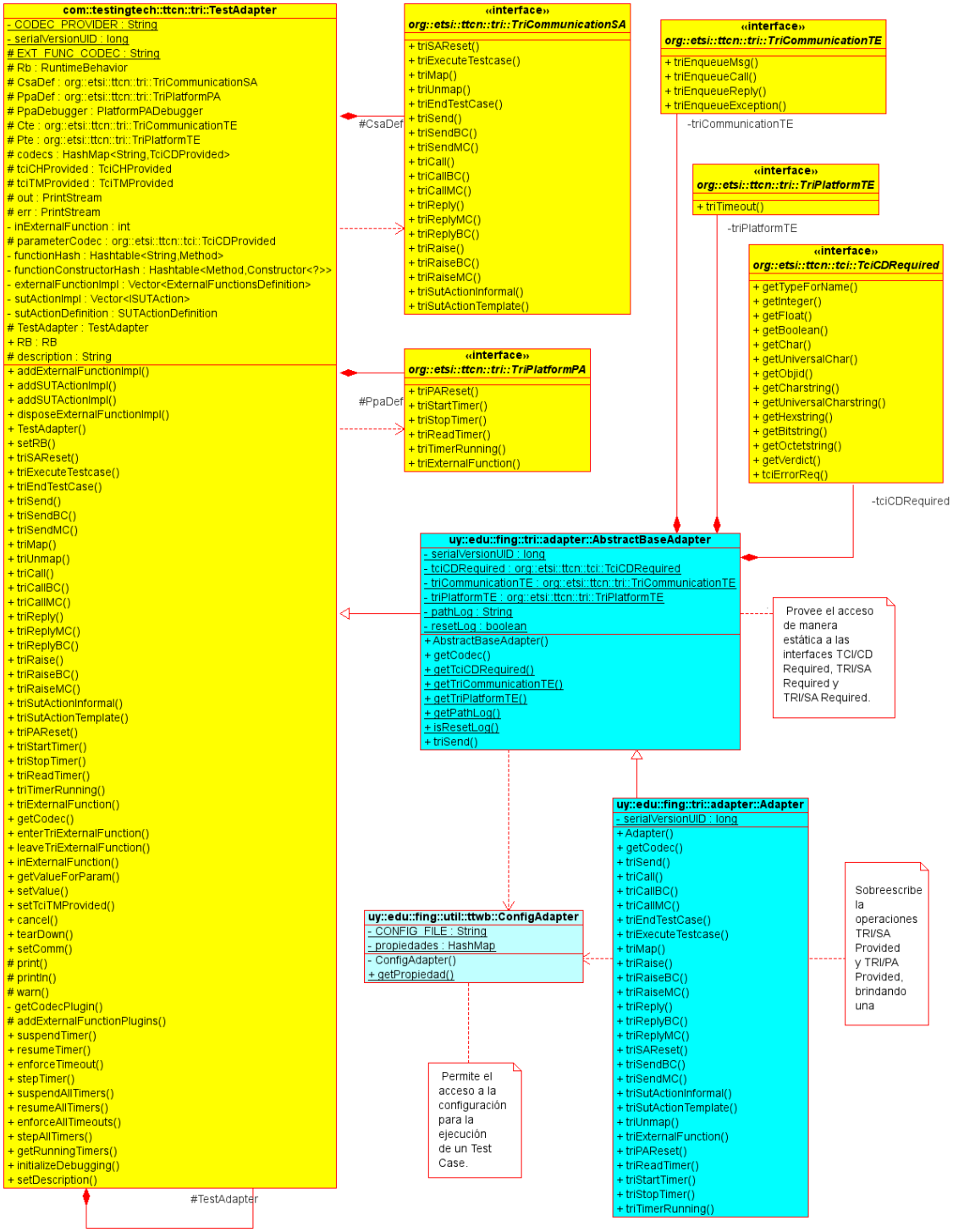


Figura 4.3: Diagrama de clases para el Adapter.

De forma similar a lo realizado en el diagrama de la Figura 4.2, los detalles de las API están accesibles en la referencia [13].

Tomando en cuenta lo mencionado con respecto a la SA y PA, se agregan las clases `uy.edu.fing.tri.adapter.AbtractBaseAdapter` y `uy.edu.fing.tri.adapter.Adapter`, las cuales participaran en la interconexión entre módulos implementados en distintos lenguajes.

La clase `uy.edu.fing.tri.adapter.AbtractBaseAdapter` extenderá a la clase `com.testingtech.ttcn.tri.TestAdapter` y la clase `uy.edu.fing.tri.adapter.Adapter` extenderá de la clase `uy.edu.fing.tri.adapter.AbtractBaseAdapter`, de esta manera se heredan las funcionalidades brindadas por el TTwb; ambas están relacionadas con la entidad encargada del manejo para la configuración (clase `uy.edu.fing.util.ttwb.ConfigAdapter`) de la ejecución de un Test Case, como se muestra en la Figura 4.3.

En la clase `uy.edu.fing.tri.adapter.AbtractBaseAdapter`, se incorpora al configuración necesaria para el manejo de la traza a nivel del conector y se sobrescribe el método `getCodec()` para de esta manera lograr la parametrización de la clase CoDec a ser utilizada. Además, incorpora propiedades para el acceso a la interfaces TCI/CD Required, TRI/SA Required y TRI/PA Required, necesarias al momento de las invocaciones desde la plataforma C/C++, las cuales completan el ciclo de interacciones entre los módulos existentes a través del conector JNI.

En el siguiente nivel de la jerarquía se tiene la clase concreta `uy.edu.fing.tri.adapter.Adapter`, la cual permite que se puedan sobrescribir los métodos de las interfaces TRI/SA Provided y TRI/PA Provided de manera nativa, logrando que de manera transparente se invoquen de Java los métodos implementados por el T3DEvKit en C/C++. Esta clase también es responsable de acceder a la configuración para acceder al conector JNI de forma dinámica.

Dentro del conector JNI, se cuenta con un conjunto de operaciones correspondientes a la clase Adapter Java con métodos nativos. Dando lugar al diseño de un módulo C que implementa las mismas.

La responsabilidades del módulo Adapter en lenguaje C, con respecto a tratamiento de las estructuras de datos, son las mismas que las del módulo Codec, mencionadas en el apartado 4.2..

4.4. Integración para las interfaces Provided

Las interfaces provistas por el compilador son publicadas por la clase `uy.edu.fing.tri.adapter.AbtractBaseAdapter`, como se mostró en diagrama de la Figura 4.3.

Revisando las funciones requeridas por el T3DevKit, es decir, viendo las invocaciones a la interfaces Provided desde C/C++, tenemos que poder acceder a la implementación Java desarrollada por TestingTech. Esto nos lleva a generar un conjuntos módulos C, uno por cada interfaz Java. Se definen cabezales de operaciones

que representan a las interfaces Java TCI/CD Required, TRI/SA Required y TRI/PA Required.

Estos módulos son responsables de traducir los objetos de intercambio con la T3DevLib, o sea, estructuras C/C++ a objetos JNI, para luego a través de las interfaces JNI acceder a las funcionalidades Java.

Además de dejar disponibles las operaciones TCI/CD Required, es necesario contar con un módulo C adicional, que permita crear valores TCI y acceder a sus atributos (carga y recuperación de sus propiedades).

De esta manera, los módulos C/C++ de la T3DevLib son capaces de invocar la funcionalidades provistas para comunicarse con el TE de manera transparente, sin conocer que estas funcionalidades pertenecen a módulos Java.

4.5. Diseño del módulo de Utilidades

Fue necesario separar en un módulo diferente las funcionalidades comunes a ser utilizadas por el conector, promoviendo la mantenibilidad, escalabilidad y re-utilización de las mismas.

Hay dos grupos bien definidos de funciones:

- las de acceso a las interfaces JNI y las de creación de objectos JNI.
- las que permiten el manejo de los archivos con la traza referente a una o varias ejecuciones de un Test Case.

4.5.1. Utilidades JNI

Se detecta que la funcionalidades de acceso al entorno JVM, son requeridas por la gran mayoría del los módulos que integran el conector.

Para cumplir con la inclusiones de estas funcionalidades de manera simple y con un fácil re-uso de las mismas, se diseño dentro de un módulo de utilidades un conjunto de cabezales con firmas dinámicas (orden en los parámetros, pero cantidad no definida a priori). Esto permite una fácil correspondencia con la interfaces y objetos JNI [10][18].

4.5.2. Utilidades de Logging

El acceso a archivos que guardan la traza se considera muy importante a la hora del análisis, esta traza corresponde a las secuencias de interacción entre las entidades que se relacionan en la ejecución de un Test Case.

Para esto, se definió dentro del módulo de utilidades un conjunto de declaraciones, que se adoptan en todos los módulos integrantes de conector, permitiendo escribir y almacenar ciertos datos relevantes a la hora de la invocaciones a las funciones a ambos lados de la plataforma dual.

Con esto se logra complementar las funcionalidades de Logging provistas por el TTwb y el T3DevKit (estas pueden ser habilitas o des-habilitadas, sobrescribiendo su configuración).

4.6. Ventajas del diseño preliminar

El diseño propuesto facilita el re-uso de componentes de software, en particular re-utilización de los módulos C actualmente funcionando, esto permite evitar, por ejemplo generar una nueva implementación del T3DevKit completamente en Java; lo cual implicaría mayor costo de desarrollo y mantenimiento.

Con esta solución se mantiene una única pieza de software. Si se hubiera optado por una solución que implicara traducción de código o re-implementación de módulos, se aumentarían las dificultades a la hora del mantenimiento, dado que, los impactos de los cambios deberían reflejarse en ambos módulos.

La idea principal es disponer de un generador de CoDecs que pueda ser utilizado desde la plataforma Java o C/C++ indistintamente, por lo cual se aprovecha al máximo el hecho de que las interfaces están bajo un estándar. Esto implica, simplicidad al momento de la integración e interoperación entre módulos.

Solamente es requerido mantener una pieza de software, pues no es un costo nuevo el mantenimiento del T3DevKit. Por otra parte, el conector es algo estático respecto a la traducción realizada en cada compilación de TTCN-3. Luego de validado no deberá ser modificado hasta que cambie la definición de las API TCI y/o TRI. La opción de portar completamente a Java el T3DevKit, es decir, que traduzca código TTCN-3 a Java directamente es más costosa a la hora del mantenimiento, pues duplica todas las tareas. Aunque que existiera una base común con el T3DevKit, casi todo debería realizarse dos veces. Consideramos que la solución implementada es la más eficiente desde el punto de vista del mantenimiento, si bien, deberemos ser cuidadosos para lograr un nivel de desempeño aceptable.

4.7. Problemáticas

Es importante determinar, que representación para los tipos de datos utilizar y como generarlos en la porción correspondiente al conector, buscando la simplicidad a la hora de la implementación.

El estudio detallado de la correspondencia entre la funciones Java y C, para definir correctamente las invocaciones necesarias (tal vez algunas de ellas solo deben ser un stub, sólo con el objetivo de cumplir con las firmas de la interfaces) y la localización de las mismas.

Se debe realizar un revisión sobre el diseño del T3Devkit, estudiando las tecnologías utilizadas para su desarrollo (por ejemplo: Automake [17]), para luego comprobar que la adaptaciones realizadas sean la necesarias y que no tengan impacto en las funcionalidades ya existentes.

Se debe identificar cuales de las interfaces propias de TT, podrían no ser integradas y cual es el costo de "suavizar" la no estandarización de ciertas interfaces y si esto fuese posible.

Ante la selección de la tecnología JNI para la integración de módulos, es necesario comprender la arquitectura y su diseño, analizando posibles limitaciones, que no permitieran cumplir con los objetivos trazados en el diseño del conector.

5. Implementación del conector

En esta sección se hará mención a los detalles de implementación de clases Java, así como, también de módulos C involucrados, mostrando donde y como se invoca al T3DevKit y sus llamadas hacia la API Java.

Se mostraran las dependencias ente bibliotecas y módulos; además de describir las funciones utilitarias necesarias en la implementación.

5.1. Correspondencias TCI/CD entre Java y C

Para interactuar con T3DevKit se utilizan las interfaces de Go4IT, donde `encode()` y `decode()` es la parte "Provided" del API, y lo que se le debe brindar es la parte "Required" (dada por TT en Java), con esto se simula un Test System en lenguaje C. A este nivel, se deben implementar los métodos "Required" en C, pero que invoquen a través de JNI a los "Required" de TT, ese es el rol del wrapper JNI [12].

A nivel de implementación, se tiene una interfaz Java para comunicarse con TT y que TT interprete que se tiene un CoDec Java; y viceversa, o sea, tener una interfaz C, para que el CD interprete que se comunica con un Test System en C.

5.1.1. Required API

La Tabla 5.1 muestra la correspondencia de las funciones para el manejo de valores TCI entre los dos lenguajes de plataforma. Para ver mas detalles sobre las firmas de los métodos es necesario dirigirse a las secciones 8 y 9 de la referencia [12].

Esta operaciones son las que permiten el manejo de TCI Values a ser enviados por SUT en sus respuestas.

Definiciones Java	Definiciones C

`public interface TciCDRequired {`
`  public Type getTypeForName (...);`
`  public Type getInteger ();`
`  public Type getFloat ();`
`  public Type getBoolean ();`
`  public Type getChar ();`
`  public Type getUniversalChar ();`
`  public Type getObjid ();`
`  public Type getCharstring ();`
`  public Type getUniversalCharstring ();`
`  public Type getHexstring ();`
`  public Type getBitstring ();`
`  public Type getOctetstring ();`
`  public Type getVerdict ();`
`  public void tciErrorReq (...);`
`}`
	`Type getTypeForName (...);`
	`Type getInteger ();`
	`Type getFloat ();`
	`Type getBoolean ();`
	`Type getChar ();`
	`Type getUniversalChar ();`
	`Type getObjid ();`
	`Type getCharstring ();`
	`Type getUniversalCharstring ();`
	`Type getHexstring ();`
	`Type getBitstring ();`
	`Type getOctetstring ();`
	`Type getVerdict ();`
	`void tciErrorReq(...);`

Tabla 5.1: Correspondencia de Interfaces TCI/CD Required entre Java y C.

5.1.2. Provided API

La Tabla 5.2 muestra la correspondencia de las funciones para el manejo la codificación y de-codificación de tipos TTCN-3 entre los dos lenguajes de plataforma. Para ver mas detalles sobre las firmas de los métodos es necesario dirigirse a las secciones 8 y 9 de la referencia [12].

Definiciones Java	Definiciones C

`public interface TciCDProvided  {`
`  public Value decode (...);`
`  public TriMessage encode (...);`
`}`
	`TciValue tciDecode(...);`
	`BinaryString tciEncode(...);`

Tabla 5.2: Correspondencia de Interfaces TCI/CD Provided entre Java y C.

5.2. Correspondencias TRI/SA entre Java y C

Para interactuar con T3DevKit se utilizan las interfaces de Go4IT, donde `triSAReset()`, `triSend()`, `triMap()`, `triExecuteTestcase()`, etc., es la parte "Provided" del API, y lo que se le debe brindar es la parte "Required" (dada por TT en Java), con esto se simula un Test System en lenguaje C.

A este nivel, se deben implementar los métodos "Required" en C, pero que invoquen a través de JNI a los "Required" de TT, ese es el rol del wrapper JNI [13].

A nivel de implementación, se tiene una interfaz Java para comunicarse con TT y que TT interprete que se tiene un Adapter Java; y viceversa, o sea, tener una interfaz C, para que el SA interprete que se comunica con un Test System en C.

5.2.1. Required API

La Tabla 5.3 muestra la correspondencia de las funciones para el manejo de las respuestas a mensajes enviados por el TE, entre los dos lenguajes de plataforma. Para ver mas detalles sobre las firmas de los métodos es necesario dirigirse a las secciones 8 y 9 de la referencia [13].

Definiciones Java	Definiciones C

`public interface TriCommunicationTE {`
`  public void triEnqueueMsg(...);`
`  public void triEnqueueCall(...);`
`  public void triEnqueueReply(...);`
`  public void triEnqueueException(...);`
`}`
	`void triEnqueueMsg(...);`
	`void triEnqueueCall(...);`
	`void triEnqueueReply(...);`
	`void triEnqueueException(...);`

Tabla 5.3: Correspondencia de Interfaces TRI/SA Required entre Java y C.

5.2.2. Provided API

La Tabla 5.4 muestra la correspondencia de las funciones para el manejo de la comunicación con el Test System, entre los dos lenguajes de plataforma. Para ver mas detalles sobre las firmas de los métodos es necesario dirigirse a las secciones 6 y 7 de la referencia [13].

Definiciones Java	Definiciones C

`public interface TriCommunicationSA {`
`  public TriStatus triSAReset();`
`  public TriStatus triExecuteTestcase(...);`
`  public TriStatus triMap(...);`
`  public TriStatus triUnmap(...);`
`  public TriStatus triEndTestCase();`
`  public TriStatus triSend(...);`
`  public TriStatus triSendBC(...);`
`  public TriStatus triSendMC(...);`
`  public TriStatus triCall(...);`
`  public TriStatus triCallBC(...);`
`  public TriStatus triCallMC(...);`
`  public TriStatus triReply(...);`
`  public TriStatus triReplyMC(...);`
`  public TriStatus triReplyBC(...);`
`  public TriStatus triRaise(...);`
`  public TriStatus triRaiseBC(...);`
`  public TriStatus triRaiseMC(...);`
`  public TriStatus triSutActionInformal(...);`
`  public TriStatus triSutActionTemplate(...);`
`}`
	`TriStatus triSAReset();`
	`TriStatus triExecuteTestCase(...);`
	`TriStatus triMap(...);`
	`TriStatus triUnmap(...);`
	`TriStatus triEndTestCase();`
	`TriStatus triSend(...);`
	`TriStatus triSendBC(...);`
	`TriStatus triSendMC(...);`
	`TriStatus triCall(...);`
	`TriStatus triCallBC(...);`
	`TriStatus triCallMC(...);`
	`TriStatus triReply(...);`
	`TriStatus triReplyBC(...);`
	`TriStatus triReplyMC(...);`
	`TriStatus triRaise(...);`
	`TriStatus triRaiseBC(...);`
	`TriStatus triRaiseMC(...);`
	`TriStatus triSUTActionInformal(...);`

Tabla 5.4: Correspondencia de Interfaces TRI/SA Provided entre Java y C.

Se debe notar que, en la correspondencia para Java aparece la función `triSutActionTemplate()` aunque la misma está deprecada; mientras que, esto en la correspondencia para C ya se contempló, por lo tanto, no se incluye este método.

5.3. Correspondencias TRI/PA entre Java y C

Para interactuar con T3DevKit se utilizan las interfaces de Go4IT, donde `triPAReset()`, `triStartTimer()`, `triStopTimer()`, etc., es la parte "Provided" del API, y lo que se le debe brindar es la parte "Required" (dada por TT en Java), con esto se simula un Test System en lenguaje C.

A este nivel, se deben implementar los métodos "Required" en C, pero que invoquen a través de JNI a los "Required" de TT, ese es el rol del wrapper JNI [13].

A nivel de implementación, se tiene una interfaz Java para comunicarse con TT y que TT interprete que se tiene un Adapter Java; y viceversa, o sea, tener una interfaz C, para que el SA interprete que se comunica con un Test System en C.

5.3.1. Required API

La Tabla 5.5 muestra la correspondencia de las funciones para el manejo de la espiración de temporizadores, entre los dos lenguajes de plataforma. Para ver mas detalles sobre las firmas de los métodos es necesario dirigirse a las secciones 6 y 7 de la referencia [13].

Definiciones Java	**Definiciones C**

`public interface TriPlatformTE {`
`  public void triTimeout(...);`
`}`
	`void triTimeout(...);`

Tabla 5.5: Correspondencia de Interfaces TRI/PA Required entre Java y C.

5.3.2. Provided API

La Tabla 5.6 muestra la correspondencia de las funciones para el manejo y consulta de temporizadores, entre los dos lenguajes de plataforma. Para ver mas detalles sobre las firmas de los métodos es necesario dirigirse a las secciones 6 y 7 de la referencia [13].

Definiciones Java	**Definiciones C**

`public interface TriPlatformPA {`
`  public TriStatus triPAReset();`
`  public TriStatus triStartTimer(...);`
`  public TriStatus triStopTimer();`
`  public TriStatus triReadTimer(...);`
`  public TriStatus triTimerRunning(...);`
`  public TriStatus triExternalFunction(...);`
`}`
	`TriStatus triPAReset();`
	`TriStatus triStartTimer(...);`
	`TriStatus triStopTimer(...);`
	`TriStatus triReadTimer(...);`
	`TriStatus triTimerRunning(...);`
	`TriStatus triExternalFunction(...);`

Tabla 5.6: Correspondencia de Interfaces TRI/PA Provided entre Java y C.

5.4. Definiciones C para el manejo de TCI Values

Los objetos de trasferencia y operaciones sobre los mismos, utilizados por la API C para ejecutar las funciones de CoDec, están definidos por los TCI Data y las interfaces TCI Values respectivamente; esto aparece descrito en las secciones 9.2 y 9.5 de la referencia [12].

5.4.1. TCI Data

En las tablas 5.7 y 5.8, se muestran las estructuras de datos C que representan a los valores y tipos TCI.

Definiciones C de TCI Data

```
typedef struct _TciValue {
   JNIEnv* env;
   jobject obj; } * TciValue;

typedef struct _TciType {
   JNIEnv* env; jobject obj; } * TciType;

typedef struct _TciValueTemplate
   { int dummy; } * TciValueTemplate;
typedef struct _TciNonValueTemplate
   { int dummy; } * TciNonValueTemplate;

typedef TciValue TciVerdictValue;
typedef TciValue VerdictValue;

typedef QualifiedName
TciModuleParameterIdType;

typedef enum {
  inactiveC,
  runningC,
  stoppedC,
  killedC
} ComponentStatus;

typedef enum {
  runningT,
  inactiveT,
  expiredT
} TimerStatus;

typedef TriStatus TciStatus;

#define TCI_OK        TRI_OK
#define TCI_Error     TRI_Error

typedef QualifiedName TciModuleIdType;
typedef QualifiedName TciTestCaseIdType;

typedef struct TciModuleParameterType {
  String parName;
  TciValue defaultValue;
} TciModuleParameterType;
```

```
typedef enum {
  TCI_IN_PAR = 0,
  TCI_INOUT_PAR = 1,
  TCI_OUT_PAR = 2
} TciParameterPassingModeType;

typedef struct
TciModuleParameterListType {
  long int length;
  TciModuleParameterType *modParList;
} TciModuleParameterListType;

typedef struct TciParameterType {
  String parName;
  TciParameterPassingModeType
parPassMode;
  TciValue parValue;
} TciParameterType;

typedef struct TciParameterListType {
  long int length;
  TciParameterType *parList;
} TciParameterListType;

typedef struct TciParameterTypeListType
{
  long int length;
  TciType *parList;
} TciParameterTypeListType;

typedef struct TciTestCaseIdListType {
  long int length;
  QualifiedName *idList;
} TciTestCaseIdListType;
```

Tabla 5.7: Estructuras C para TCI Data (parte 1).

Definiciones C de TCI Data

```
typedef enum {
  TCI_ADDRESS_TYPE,
  TCI_ANYTYPE_TYPE,
  TCI_BITSTRING_TYPE,
  TCI_BOOLEAN_TYPE,
  TCI_CHAR_TYPE,
  TCI_CHARSTRING_TYPE,
  TCI_COMPONENT_TYPE,
  TCI_ENUMERATED_TYPE,
  TCI_FLOAT_TYPE,
  TCI_HEXSTRING_TYPE,
  TCI_INTEGER_TYPE,
  TCI_OBJID_TYPE,
  TCI_OCTETSTRING_TYPE,
  TCI_RECORD_TYPE,
  TCI_RECORD_OF_TYPE,
  TCI_SET_TYPE,
  TCI_SET_OF_TYPE,
  TCI_UNION_TYPE,
  TCI_UNIVERSAL_CHAR_TYPE,
  TCI_UNIVERSAL_CHARSTRING_TYPE,
  TCI_VERDICT_TYPE
} TciTypeClassType;

typedef enum {
  TCI_CTRL_COMP,
  TCI_MTC_COMP,
  TCI_PTC_COMP,
  TCI_SYS_COMP,
  TCI_ALIVE_COMP
} TciTestComponentKindType;

typedef struct TciValueDifference {
  TciValue val;
  TciValueTemplate tmpl;
  String desc;
} TciValueDifference;
```

(continuación)

```
typedef QualifiedName TciBehaviourIdType;

typedef struct TciValueDifferenceList {
  long int length;
  TciValueDifference* diffList;

} TciValueDifferenceList;

const int TCI_VERDICT_NONE =   0;
const int TCI_VERDICT_PASS =   1;
const int TCI_VERDICT_INCONC = 2;
const int TCI_VERDICT_FAIL =   3;
const int TCI_VERDICT_ERROR =  4;

typedef struct TciObjidElemValue {
  char* elem_as_ascii;
  long int elem_as_number;
  void* aux;
} TciObjidElemValue;

typedef struct TciObjidValue {
  long int length;
  TciObjidElemValue *elements;
} TciObjidValue;


typedef struct TciCharStringValue {
  unsigned long int length;
  char* string;
} TciCharStringValue;

typedef unsigned char TciUCValue[4];

typedef struct TciUCStringValue {
  unsigned long int length;
  TciUCValue *string;
} TciUCStringValue;
```

Tabla 5.8: Estructuras C para TCI Data (parte 2).

5.4.2. TCI Value

En las tablas 5.9, 5.10 y 5.11, se presentan las operaciones definidas en las interfaces C para la creación acceso a valores y tipos TCI.

Funciones C de TCI Value

```
/* Type */
TciModuleIdType
tciGetDefiningModule(TciType inst);
String tciGetName(TciType inst);
TciTypeClassType tciGetTypeClass
(TciType inst);
TciValue tciNewInstance(TciType inst);
String tciGetTypeEncoding(TciType inst);
String* getTypeExtension(TciType inst);
String
tciGetTypeEncodingVariant(TciType
inst);
```

```
/* Value */
String tciGetValueEncoding(TciValue inst);
String tciGetValueEncodingVariant(TciValue
inst);
TciType tciGetType(TciValue inst);
Boolean tciNotPresent(TciValue inst);
```

Tabla 5.9: Operaciones C para TCI Value (parte 1).

Funciones C de TCI Value

```
/* IntegerValue */
String tciGetIntAbs(TciValue inst);
unsigned long int
tciGetIntNumberOfDigits(TciValue inst);
Boolean tciGetIntSign(TciValue inst);
char tciGetIntDigit(TciValue inst,unsigned
long int position);
void tciSetIntAbs(TciValue inst, String
value);
void tciSetIntNumberOfDigits(TciValue
inst,unsigned long int dig_num);
void tciSetIntSign(TciValue inst,Boolean
sign);
void tciSetIntDigit(TciValue inst,unsigned
long int position,char digit);

/* FloatValue */
double tciGetFloatValue(TciValue inst);
void tciSetFloatValue(TciValue inst, double
value);

/* BooleanValue */
Boolean tciGetBooleanValue(TciValue inst);
void tciSetBooleanValue
(TciValue inst, Boolean value);

/* ObjidValue */
TciObjidValue tciGetTciObjidValue(TciValue
inst);
void tciSetObjidValue(TciValue
inst,TciObjidValue value);

/* CharstringValue */
TciCharStringValue
tciGetCStringValue(TciValue inst);
void tciSetCStringValue(TciValue inst,
TciCharStringValue value); /* was:
TciCharString */
char tciGetCStringCharValue(TciValue inst,
long int position);
void tciSetCStringCharValue(TciValue
inst,long int position,char value);
unsigned long int tciGetCStringLength
(TciValue inst);
void tciSetCStringLength (TciValue
inst,unsigned long int len);

/* UniversalCharstringValue */
TciUCStringValue
tciGetUCStringValue(TciValue inst);
void tciSetUCStringValue(TciValue inst,
TciUCStringValue value);
void tciGetUCStringCharValue(TciValue inst,
unsigned long int position,TciUCValue
result);
void tciSetUCStringCharValue(TciValue
inst,unsigned long int position,TciUCValue
value);
unsigned long int
tciGetUCStringLength(TciValue inst);
void tciSetUCStringLength(TciValue
inst,unsigned long int len);
```

(continuación)

```
/* BitstringValue */
String tciGetBStringValue(TciValue inst);
void tciSetBStringValue(TciValue inst,
String value);
int tciGetBStringBitValue(TciValue inst,
long int position);
void tciSetBStringBitValue(TciValue
inst,unsigned long int position,int value);
unsigned long int
tciGetBStringLength(TciValue inst);

/* HexstringValue */
String tciGetHStringValue(TciValue inst);
void tciSetHStringValue(TciValue inst,
String value);
int tciGetHStringHexValue(TciValue inst,
unsigned long int position);
void tciSetHStringHexValue(TciValue
inst,unsigned long int position,int value);
long int tciGetHStringLength(TciValue
inst);
void tciSetHStringLength(TciValue
inst,unsigned long int len);

/* OctetstringValue */
String tciGetOStringValue(TciValue inst);
void tciSetOStringValue(TciValue inst,
String value);
int tciGetOStringOctetValue(TciValue inst,
unsigned long int position);
void tciSetOStringOctetValue(TciValue
inst,unsigned long int position,int value);
unsigned long int
tciGetOStringLength(TciValue inst);
void tciSetOStringLength(TciValue
inst,unsigned long int len);

/* RecordValue */
TciValue tciGetRecFieldValue(TciValue
inst,String fieldName);
void tciSetRecFieldValue(TciValue
inst,String fieldName,TciValue value);
char** tciGetRecFieldNames(TciValue inst);
void setFieldOmitted(String fieldName);
```

Tabla 5.10: Operaciones C para TCI Value (parte 2).

Funciones C de TCI Value

```
/* RecordOfValue */
TciValue tciGetRecOfFieldValue(TciValue
inst,unsigned long int position);
void tciSetRecOfFieldValue(TciValue
inst,unsigned long int position,TciValue
value);
void tciAppendRecOfFieldValue(TciValue
inst,TciValue value);
TciType tciGetRecOfElementType(TciValue inst);
unsigned long int tciGetRecOfLength(TciValue
inst);
void tciSetRecOfLength(TciValue inst,unsigned
long int len);
TciValue tciGetUnionVariant(TciValue
inst,String variantName);
void tciSetUnionVariant(TciValue inst,String
variantName,TciValue value);
String tciGetUnionPresentVariantName(TciValue
inst);
char** tciGetUnionVariantNames(TciValue inst);
String tciGetEnumValue(TciValue inst);
void tciSetEnumValue(TciValue inst,String
enumValue);
```

(continuación)

```
/* VerdictValue */
int tciGetVerdictValue(TciValue inst);
void tciSetVerdictValue(TciValue inst,
int verdict);

/* AddressValue */
TciValue tciGetAddressValue(TciValue
inst);
void tciSetAddressValue(TciValue inst,
TciValue value);
```

Tabla 5.11: Operaciones C para TCI Value (parte 3).

5.5. Detalles sobre la biblioteca de Utilidades

Como se mencionó en la sección 4.5., se cuenta con un módulo de utilidades comunes, estas funcionalidades se agrupan en una biblioteca estática C (extensión `.a` en Unix y `.lib` en Windows), de esta manera se reutiliza el código, sin tener que implementar las mismas rutinas en otros módulos relacionados, estas se escriben una vez y se hace referencia a ellas desde las aplicaciones que necesiten estas funcionalidades.

5.5.1. Implementación de funcionalidades JNI

Este módulo incluye las interfaces JNI para poder implementar el acceso a la instancia de la JVM que realiza las invocaciones a los métodos nativos.

Se definen en el .h de este módulo, variables para mantener el contexto al momento de la correspondencia con la interfaces Required, en particular, es posible guardar una referencia a la clase Adapter Java y sus métodos estáticos de acceso a las interfaces TriCommunicationTE, TriPlatformTE y TciCDRequired. Se implementa de esta forma, dado que para la tecnología JNI los accesos a atributos y procedimientos estáticos es mas simple y eficiente [10][18].

Se implementan funciones básicas para la interacción con la JVM: crear una nueva instancia, recuperar la instancia invocadora (pudiendo asociarse a un hilo en ejecución), liberar una instancia y desasociar hilos en ejecución.

Se cuenta con la implementación de las búsquedas necesarias para realizar invocaciones a la JVM y creación de objetos JNI. Es importa recuperar la clase

(`jclass`) a partir de un objeto (`jobject`) y recuperar el identificador de un método (`jmethodID`) a partir de la clase (`jclass`) y su firma.

En base a las funciones mencionadas anteriormente, se "enmascaran" las invocaciones a las funciones JNI para la creación de objetos, arrays (por ejemplo: de bytes, chars, objects), transformación de objetos `jstring` a cadenas de caracteres y viceversa; y llamadas a funciones y procedimientos Java (comunes y estáticos). Para el caso de las funciones algunos de los objetos JNI de retorno son: `jobject`, `jboolean`, `jint`, `jchar` y `jfloat` [19].

Cabe mencionar, que los métodos definidos en esta parte de la biblioteca son utilizados tanto para la invocaciones en dirección Java → C/C++, así como también, para la invocaciones en dirección C/C++ → Java. Por lo que, la utilización principal de la biblioteca es desde los módulos del conector, pero dada la generalidad de la misma, es posible que se utilizada por otro tipo de módulo.

5.5.2. Implementación de funcionalidades de Logging

La biblioteca de utilidades también provee de métodos para la escritura de archivos para el mantenimiento de la traza de ejecución a nivel del conector, estas funcionalidades no tienen vinculación alguna con la TL (interfaz que provee la información de la traza sobre la ejecución de un test, incluyendo también la información provista por las sentencias de log en el código TTCN-3 y brinda la capacidad de logging para todos los elementos en la arquitectura del Test System, como puede verse en la Figura 2.1).

La utilidad de este módulo de logging es poder saber que correspondencias se ejecutaron dentro del conector y poder realizar el seguimiento de invocaciones hacia ambas plataformas, por lo que, en lo mencionado se marcan las diferencias con la implementación de las interfaces TCI/TL.

Se definen e implementan funciones para crear y/o abrir el archivo de Logging, imprimir lineas y poder realizar el cierre del mismo [20][21].

Al momento de realizar las impresiones se realiza una marca de tiempo, donde se determina la fecha y hora del evento en la función invocadora.

De la misma forma que las utilidades JNI, las funciones implementadas en esta parte de la biblioteca son invocadas en ambas direcciones para la plataforma dual y también son suficientemente genéricas como para ser invocadas desde otros módulos.

5.6. Detalles sobre la conexión Java → C/C++

En base al diseño explicado en los apartados 4.2. y 4.3., se generan las clases Java que implementan las interfaces en forma nativa y sus correspondientes en C (las cuales "enmascaran" la invocaciones al T3DevKit).

Este grupo de funciones se publicada en una biblioteca dinámica C (extensión `.so` en Unix y `.dll` en Windows), las subrutinas de la misma, son cargadas en las aplicaciones en tiempo de ejecución y se mantienen como archivos independientes separados del ejecutable para el programa principal, teniendo la ventaja de que varios

programas pueden compartir las mismas librerías. Para este proyecto, contar con una biblioteca dinámica, hace posible que la misma pueda ser cargada desde clases Java [19] [22].

5.6.1. Implementación del Codec

En Java se implementa una clase `Codec` que extiende de la clase `AbstractBaseCodec` de TT. La clase `Codec` se encarga de realizar la implementación de la interfaz TCI/CD Provided, esta implementación como se mencionó en las descripciones del diseño, quedan declaradas como `native`, esto da lugar al módulo C `Codec` (correspondiente a la implementación de la clase `Codec` Java), el cual incluye los cabezales generados a partir de los métodos nativos se encarga de implementar los mismos. En concreto, en cada uno de los métodos, se convierten los parámetros a estructuras de datos C, para luego realizar las llamadas correspondientes al T3DevKit.

Concretamente lo que se hace aquí, es tomar los valores TCI Java y convertirlos en valores TCI C/C++, para luego invocar a la operaciones de codificación/de-codificación, tomando los valores de retorno de la funciones C/C++ y transformándolos en los correspondientes objetos Java de retorno.

El módulo C incluye y utiliza las funcionalidades de acceso a las interfaces JNI y además, invoca a los métodos de Logging para generar la traza de ejecución de sus métodos.

5.6.2. Implementación del Adapter

En Java se implementan dos clases: `AbtractBaseAdapter` y `Adapter`. La clase `AbtractBaseAdapter` extiende de la clase `TestAdapter` de TT. En el siguiente nivel de la jerarquía la clase `Adapter` extiende de la clase `AbtractBaseAdapter` y se encarga de la implementación de las interfaces TRI/SA Provided y TRI/SA Provided, las mismas son declaradas como `native` siguiendo el diseño descrito.

La clase `AbtractBaseAdapter` sobrescribe el método `getCodec()`, realizando la localización de la clase `Codec` especificada para la ejecución del test. Esta porción de código es la indicada para acceder al RB (Runtime Behavior) del TTwb, que es una interfaz que se define como principal punto de entrada al comportamiento en tiempo de ejecución del TTthree (clases Java del compilador y encargadas del T3RTS), donde las entidades del Test System están almacenadas y pueden ser obtenidas. La clase `AbtractBaseAdapter` mediante el RB obtiene y publica las implementaciones de las interfaces TciCDRequired, TriCommunicationTE y TriPlatformTE (estas dos últimas, correspondientes a la parte Required de la TRI), permitiendo el acceso estático a las mismas, para que el conector JNI las acceda. También guarda la configuración (utilización de la clase `ConfigAdapter` detallada en el apartado 5.9.) para el manejo de la traza a nivel del conector (se cargan las propiedades para la ruta al archivo de Logging y la modalidad de escritura, o sea, si se reinicia o no el contenido del archivo).

En tanto, la clase `Adapter`, además de definir la implementación de la interfaces como native, dando lugar al módulo C `Adapter`, carga la biblioteca dinámica C/C++ en

base a la configuración (utilización de la clase `ConfigAdapter` detallada en el apartado 5.9.).

El módulo `Adapter` en C (correspondiente a la implementación de la clase `Codec` Java), el cual incluye los cabezales generados a partir de los métodos nativos se encarga de implementar los mismos. El trabajo de este módulo es tomar los parámetros (objetos Java) y transformarlos en las estructura C/C++ que representa a los objetos TRI: parámetros, listas de parámetros, puertos, componentes, direcciones, mensajes e identificadores (para timers, test cases y funciones), para luego realizar las llamadas a la funciones TRI del T3DevKit; lo valores de retorno de la funciones C/C++, son tomados para crear una instancia de la clase `TriStatus` y retornarla al TE Java.

No se implementan todas las correspondencias indicadas en el apartado 5.2.2.; las funciones para envíos de mensajes sobre componentes de forma broadcast y multicast (`triSendBC()` y `triSendMC()`), las funciones para invocación a operaciones sobre componentes de forma broadcast y multicast (`triCallBC()` y `triCallMC()`), las funciones para la el proceso de respuestas sobre componentes (`triReply()`, `triReplyBC()`, `triReplyMC()`) y los métodos raise sobre componentes (`triRaise()`, `triRaiseBC()`, `triRaiseMC()`), no son utilizadas por la T3DevLib.

Esta funcionalidades podrán ser implementadas en próximas versiones de conector, esperando también, que en siguientes versiones del T3DevKit estas métodos sean invocados.

En el análisis de los métodos para ambas plataformas, se detectó que la función `triSutActionTemplate()` esta deprecada; por lo que, en su implementación C, se retorna una instancia de la clase `TriStatus` con el valor error.

Al igual que el módulo `Codec` C, se incluyen y utilizan las funcionalidades de acceso a las interfaces JNI e invocan a los métodos de Logging.

5.7. Detalles sobre la conexión C/C++ → Java

En el apartado 4.4. se describió el diseño para permitir la invocaciones a la funciones Required desde el T3DevKit, esto da lugar a la implementación de los módulos `tciCDRequired.c`, `triSARequired.c` y `triPARequired.c`, los cuales realizan la correspondencia con las interfaces Java TCI/CD Required, TRI/SA Required, TRI/PA Required respectivamente.

También es necesario implementar el módulo `tciValues.c`, para completar las tareas de instanciación (invocaciones a constructores Java) y acceso a los atributos de TCI Values (operaciones get and set Java). Como parte de la implementación del manejo de valores TCI, se estudio como mantener referencias al entorno JNI (JNIEnv) y a la instancia de la clase Java que actúa como objeto de transferencia entre las API de codificación/de-codificación para ambas plataformas, en tal sentido, se modificaron las definiciones en la estructuras de datos para `TciValue` y `TciType` brindados por la interfaces Go4IT. Como se muestra en la Tabla 5.7, se mantienen punteros a JNIEnv (para contar en todo momento con el acceso a la interfaz nativa) y un `jobject` (para poder acceder a la instancia del objeto de transferencia) [19].

Se implementaron la gran mayoría de la funciones indicadas en: Tabla 5.9, Tabla 5.10 y Tabla 5.11, a excepción de los siguientes métodos: `tciGetUCStringValue()`, `tciSetUCStringValue()`, `tciGetUCStringCharValue()`, `tciSetUCStringCharValue()`, `tciGetAddressValue()` y `tciSetAddressValue()`. El manejo de estos valores TCI no es implementado por el T3DevKit como se menciona en el apartado 2.2.4., además en el caso `address`, tampoco es implementado por el TTwb y en el caso de `universal charstring`, es muy compleja la correspondencia entre su representación Java y la estructura de datos C, por lo que se decidió, no implementar las operaciones sobre este tipo, dejándose para trabajos futuros; de todas formas, se implementa el manejo de los tipos `object id` y `float`, esperando que en versiones futuras el T3DevKit implemente el tratamiento de estos tipos TTCN-3.

Cabe mencionar que para las funciones TRI/SA Required, no se implementan todas las correspondencias indicadas en el apartado 5.2.1., dado que las operaciones: `triEnqueueCall()`, `triEnqueueReply()` y `triEnqueueException()`, no son utilizadas por la T3DevLib, también son dejadas para ser contempladas en implementaciones futuras, esperando la evolución del T3DevKit.

Este grupo de funciones es publicada en una biblioteca estática C, haciendo posible que la misma pueda ser "enlazada" con la biblioteca generada por el T3DevKit.

5.8. Detalles sobre las invocaciones a interfaces en un Test Case

Como ya se ha mencionado a lo largo del desarrollo de este trabajo, la ejecución de un Test Case involucra una secuencia de invocaciones a las interfaces TCI y TRI. A los efectos de mostrar de una forma tangible la utilización de las interfaces implementadas y la utilidad del conector, describiremos la secuencia de invocaciones requeridas para realizar una operación abstracta de `send()` del lenguaje TTCN-3. Es importante destacar que en base a las operaciones `send()` y `receive()` se prueba un sistema reactivo con TTCN-3. Ambas operaciones son fundamentales para la aplicación de lenguaje en la práctica.

Para mostrar los detalles de dichas secuencias se tomó como ejemplo el seguimiento para la ejecución de un módulo TTCN-3 básico, la traza a nivel del wrapper obtenida se encuentra en el Apéndice E. y muestra como interviene el conector en las tareas de traducción/comunicación de los mensajes entre las herramientas integradas.

El test especificado en TTCN-3 define un tipo para la representación de los mensajes, que serán enviados a través de un puerto el cual pertenece a un componente, inmediatamente de enviado el mensajes se inicializa un temporizador para permitir la finalización de la prueba, si no se recibiese la respuesta, cuando el mismo expire. Luego de estas sentencias se realiza el llamado al `receive()` para la espera de la respuesta.

Al comienzo de la ejecución del TE se carga la biblioteca dinámica que permite la incorporación del conector, en este punto se invocan los métodos `triSAReset()` y `triPAReset()` de TRI/SA Provided y TRI/PA Provided respectivamente. Inmediatamente después es invocada la función que inicia la ejecución dentro del SA (`triExecuteTestCase()` de TRI/SA Provided).

A nivel del código TTCN-3 se ingresa a un bloque `testcase`, y en este momento se realizan las correspondencias entre los puertos virtuales y los puertos del sistema bajo prueba, mediante invocaciones a la función `triMap()` de TRI/SA Provided.

Luego dentro del código TTCN-3 se ejecuta una sentencia `send()`, lo que implica la realización de la codificación del mensaje a ser enviado, el TE llama a la operación `encode()` de TCI/CD Provided, un vez que el mensaje es codificado a una cadena de bits se invoca a la operación `triSend()` de TRI/SA Provided, en la siguiente instrucción se inicializa el temporizador llamando al método `triStartTimer()` de TRI/PA Provided, quedando luego a la espera de la respuesta, correspondiente a la sentencia `receive()` en el fuente TTCN-3.

Una vez que el SUT procesó el mensaje encola la repuesta llamando a la función `triEnqueueMsg()` de TRI/SA Required, permitiendo así que el TE acceda al mensaje de respuesta. Cuando el TE recibe la respuesta se ejecuta la de-codificación del mensaje invocando implícitamente al método `decode()` de TCI/CD Provided, para convertir la representación binaria a un valor TTCN-3, luego se compara el valor recibido con el esperado y se define el veredicto de la prueba. En el lapso correspondiente a la espera de la respuesta el TE se encarga de consultar el estado del temporizador llamando a los métodos `triTimerRunning()` y `triReadTimer()` de TRI/PA Provided

Al finalizar el testcase el TE se encarga de realzar algunas tareas como: parar el temporizador invocando al método `triStopTimer()` de TRI/PA Provided, desvincular los puertos virtuales y reales llamando a `triUnmap()` de TRI/SA Provided, y por último finalizar invocando a `triEndTestCase()` de TRI/SA Provided.

5.9. Detalles sobre la configuración

Para darle mayor flexibilidad a la ejecuciones de los test, se implementa la clase estática Java `ConfigAdapter`, encargada de leer un archivo de propiedades (`configAdapter.properties`, archivo estándar de propiedades Java, con formato orientado a líneas que contengan 'nombre_propiedad=valor_asignado' [22]), dejándolas disponibles para su lectura desde las clases encargadas de implementar el Adapter.

Esta clase provee el acceso a la configuración de un test, donde se especifican la definiciones de propiedades indicando que clase CoDec se va utilizar, la biblioteca dinámica C/C++ a ser cargada y cual sera el archivo de Logging a nivel del conector.

A continuación se enumeran y explican las propiedades incorporadas:

- *codec.package:* nombre del paquete Java que contiene a la clase CoDec a ser utilizada en el Test Case, por ejemplo: `uy.edu.fing.tci.cd.test`
- *codec.name:* nombre de la clase Java que encargada de la codificación/de-codificación a ser utilizada en el Test Case, por ejemplo: `Codec`
- *lib.path:* camino la biblioteca dinámica C/C++ a ser cargada, por ejemplo: `/usr/local/muttcn3/JNI_Java_C/lib`
- *lib.name:* nombre de la biblioteca dinámica C/C++ a ser cargada, por ejemplo: `libTTwbNativo.so`

- *log.path:* camino al archivo donde se imprimirá a traza de la ejecución de un Test Case, por ejemplo: `/tmp/muttcn3/log`
- *log.name:* nombre del archivo donde se imprimirá a traza de la ejecución de un Test Case, por ejemplo: `ttcn3Nativo.log`
- *log.reset:* opción que determina la modalidad de apertura del archivo para el manejo de la traza; en caso de que sea `true`, se reiniciará el contenido del archivo para cada una de las ejecuciones, en cualquier otro caso se imprimirá la información al final del archivo.

Dentro de la implementación del conector, también es necesario ejecutar algunas sentencias para brindar un contexto de ejecución a nivel de los módulos C/C++. Para lograr esto, se implementa un módulo `config.c`.

Aquí, se definen los métodos `JNI_OnLoad()` y `JNI_OnUnLoad()`, correspondientes a las sentencias que se ejecutan al momento de la carga y descarga de la biblioteca dinámica respectivamente [19].

En la función `JNI_OnLoad()`, se obtiene la instancia del Adapter Java, para luego recuperar y almacenar las referencias a las interfaces Provided brindadas por el compilador TTwb, también se encarga de ejecutar la funciones `triSAReset()` y `triPAReset()`, para reiniciar los contextos de SA y PA respectivamente.

Por ultimo, en la función `JNI_OnUnLoad()`, se realiza el cierre del archivo de Logging.

6. Pruebas

En esta sección se presentan las pruebas para el conector y los módulos integrados. Se comienza con las descripción de las pruebas ejecutadas para el manejo de TCI Values junto a las operaciones de codificación y de-codificación, para luego mostrar las pruebas realizadas ejecutando los Test Cases brindados por el T3DevKit.

De esta forma se está probando la traducción de tipos y valores entre las plataformas Java y C/C++, además de la correspondencia entre funciones ejecutadas a través del conector. Esto es sumamente importante ya que al probar las correspondencias entre invocaciones desde cada una de las plataformas hacia el conector se prueba la integración de las herramientas sin perdida de ninguna de sus funcionalidades actuales. De aquí se puede concluir que en la practica se está probando la interoperabilidad entre el compilador TTwb y sus entidades participantes con los módulos brindados por el T3DevKit para el desarrollo automático de CoDecs.

Es importante destacar que las pruebas realizadas están basadas en Test Cases TTCN-3 que fueron generados para realizar pruebas funcionales sobre protocolos u otras piezas de software, los cuales no tienen ninguna relación con el conector JNI desarrollado para este trabajo.

Finalmente se realiza un análisis de los resultados obtenidos mencionando la detección de algunas carencias de las herramientas y sus interfaces.

6.1. Pruebas realizadas

Se planificaron y realizaron un conjunto de pruebas incrementales. De esta forma en cada etapa se van agregando las entidades participantes en la ejecución de un Test Case, facilitando la identificación de las fallas en cada entidad. Los módulos TTCN-3 ejecutados son: `PruebaTciValue.ttcn3`, `PruebaTciCD.ttcn3`, `DNSTester.ttcn3`, `DNSTester2.ttcn3`, `MixedCoDec.ttcn3` y `RIPng`, las ejecuciones y sus resultados son detalladas en los apartados 6.1.1., 6.1.2. y 6.1.3..

Como prueba de concepto se inició el proceso de pruebas con los TCI Values. Dadas las declaraciones de tipos básicos y definidos por el usuario en un módulo TTCN-3 estos son enviados a través de un Port genérico sin esperar una respuesta, el cometido es desplegar los valores cargados en las estructuras que llegan a los módulos de prueba implementados en C. Por otra parte para validar la factibilidad de las invocaciones

desde C a Java se implementa una clase de test para TciCDRequired, dicha clase integra el módulo de test C que invoca los métodos implementados por el compilador TTwb. En resumen, esta infraestructura logra realizar la simulación de las invocaciones necesarias, hacia y desde el T3DevKit.

Luego de que fueron probadas la traducciones implementadas por el wrapper se procedió a realizar las pruebas en el primer paso de integración con el T3DevKit. Esto implica que el compilador TTwb utilizará los métodos de codificación y decodificación implementados por la T3DevLib y además que se invoque la interfaz TciCDRequired brindada por TTwb desde el T3Devkit. En esta etapa aún se siguen enviando los valores a través del Port genérico, pero en cambio se espera una respuesta en el módulo TTCN-3 que debe ser el mismo valor que se envió en la invocación.

Para la última etapa se ejecutaron los Test Case incluidos en T3Devkit. En este punto se utilizan las funciones de la clase Adapter, de esta forma se invocan desde el compilador TTwb las funciones TRI implementadas por e T3DevKit, dejando de utilizar el puerto genérico de la implementación del TTwb y desde el T3DevKit serán invocadas las funciones TRI Required brindadas por el TTwb. Con esto se completa el ciclo de invocaciones e interacciones entre las entidades participantes en los Test Cases utilizados.

6.1.1. Envío de TCI Values

Dada la importancia de los valores TCI las pruebas de intercambio de las representaciones entre ambas plataformas fueron las primeras en definirse y realizarse. El objetivo principal de estas pruebas es simular las actividades realizadas para toda la comunicación entre TTCN-3 y los adaptadores, en particular con el T3DevKit.

Para comenzar con el ciclo de pruebas lo primero que se necesitó fue un módulo TTCN-3 (`PruebaTciValue.ttcn3`) que definiera templates para cada uno de los tipos básicos y tipos definidos por el usuario (`record`, `enumerated`, `set`, `union`, `record of` y `set of`), un Port que fuese capaz de manejar estos tipos y un componente que usase el Port permitiendo la comunicación con el SUT. La implementación utilizada para la operación `triSend()` está dada por el TTwb ya que por el momento solo interesa enviar mensajes al SUT, en este caso el módulo C de test que simula al T3DevKit, para que éste despliegue las estructuras de datos generadas.

En este punto de las pruebas no se realiza codificación ni de-codificación, entonces se utiliza un CoDec especial (`TestTciCodec.java`) para realizar el test, el cual implementa las funciones `encode()` y `decode()`. En `encode()`, simplemente se invocan a la interfaces con implementación nativa `TestTciValue.java` y `TestTciCDRequired.java`, diseñadas con el fin integrar la invocaciones Java → C con el módulo de pruebas y para que se realicen las invocaciones C → Java desde el módulo de pruebas hacia las implementación brindada por el TTwb respectivamente; mientras que la función `decode()` no debe ser llamada en estas pruebas ya que no se reciben mensajes en el módulo TTCN-3.

En los módulos C se realiza la implementación nativa de los métodos. En el caso del manejo de TCI Values cada una de estas funciones crea un `TciValue` en C con la referencia al JNIEnv y el objeto Java correspondiente, para luego invocar al módulo

simulador encargado de desplegar a información contenida en las estructuras. Mientras para las invocaciones a TciCDRequired, se ejecutan pruebas para el manejo de `types` y `error` por separado; aquí nuevamente el módulo simulador toma los valores de retorno de las funciones y despliega los resultados.

Se utilizaron la herramientas de seguimiento y análisis sobre la ejecución brindadas por el TTwb en su versión del TTman para visualizar los resultados, desde el TTCN-3 Graphical Logging se exportó la Figura 6.1, donde se puede ver como fueron realizados los envíos por Port.

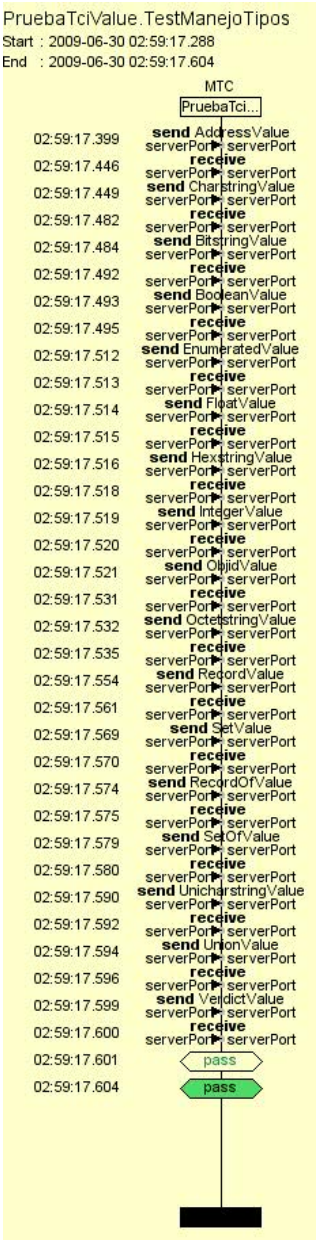


Figura 6.1: PruebaTciValue.ttcn3 (Graphical Logging)

Los tipos TTCN-3 para los cuales se prueba su envío son: `AddressValue` en la representación Java del TTwb este valor es una interfaz que no esta implementada, de todas formas se realiza el envío dejando, preparada la implementación dentro del conector, a ser completada cuando se concrete la implementación Java de la interfaz, `CharstringValue`, `BitstringValue`, `BooleanValue`, `EnumeratedValue`, `FloatValue`, `HexstringValue`, `IntegerValue`, `ObjidValue`, `OctetstringValue`, `RecordValue`, `SetValue`, `RecordOfValue`, `SetOfValue`, `UnicharstringValue`, `UnionValue` y `VerdictValue`.

En el Apéndice C. se presenta la impresión del Logging para el conector y la salida por consola Java. La consola muestra resultados `true` para la invocaciones a funciones que se han realizado con éxito, y en cualquier otro caso se muestra un mensaje con la descripción de porque la función no se ha ejecutado completamente; mientras que, el archivo de Logging, muestra el resultado del manejo de la funciones de acceso y modificación para los tipos TTCN-3 a través del conector, para cada tipo se incluye un encabezado con el nombre del mismo y a continuación los valores que han tomado sus atributos.

En la Figura 6.2 aparece la vista del TTCN-3 Textual Logging, es decir la traza de la interacción entre las entidades participantes en la ejecución del Teste Case y la vista Test Data View, mostrando el contenido de un mensaje en particular. También pueden verse las propiedades de Test Case incluyendo su veredicto e iconos que lo representan, así como también su resultado.

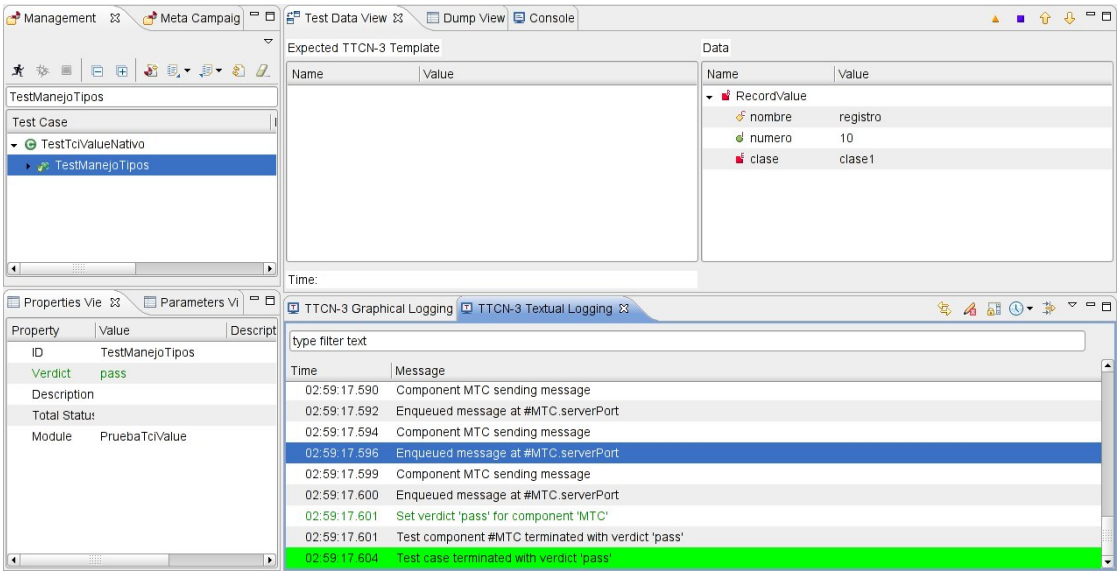


Figura 6.2: PruebaTciValue.ttcn3 (TTman)

6.1.2. Integración del Codec Nativo

El utilizar el Codec con implementación nativa, equivale a utilizar la implementación de la funciones de codificación y de-codificación dadas por el T3devKit.

Para esta etapa de las pruebas se generó un módulo TTCN-3 (`PruebaTciCD.ttcn3`) con las mismas definiciones de tipos, puertos y componentes que en el módulo `PruebaTciValue.ttcn3` descrito en el apartado anterior (6.1.1.). También en este caso la implementación utilizada para la operación `triSend()` está dada por el TTwb y en estas pruebas se utiliza desde el módulo TTCN-3 la operación `receive()`, con esto se aseguran las invocaciones a las operaciones `encode()` y `decode()` integradas con los módulos C/C++. Lo importante en esta etapa es probar la integración de la entidades participantes para la implementación de un CoDec para ambas APIs.

En resumen el módulo TTCN-3 ayuda a realizar la automatización de estas pruebas, básicamente envía una instancia de un tipo TTCN-3 (operación `send()`) y espera recibir la misma instancia (operación `receive()`), en caso de que sea diferente el veredicto no será `pass`. Con respecto a la interacción de entidades se destaca la utilización de la clase Codec, lo que permitió ajustar las operaciones `encode()` y `decode()`, realizando la comparación entre valores TTCN-3 codificados y luego de-decodificados.

Al igual que en la prueba anterior, desde el TTCN-3 Graphical Logging se exportó la Figura 6.3, donde se puede ver como fueron realizados los envíos y recepciones sobre el Port pudiéndose notar gráficamente la comparación entre el valor enviado y recibido.

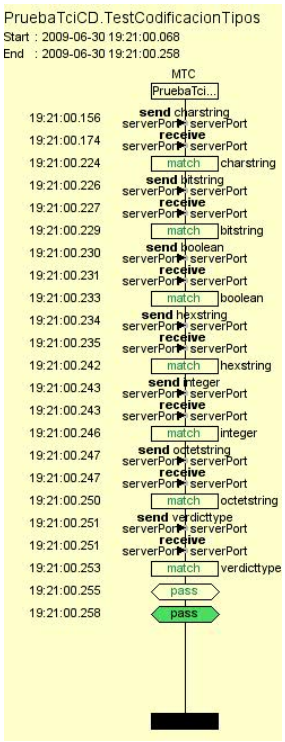


Figura 6.3: PruebaTciCD.ttcn3 (Graphical Logging)

En la Figura 6.4 aparecen la vista del TTCN-3 Textual Logging con el seguimiento de los envíos, recepciones y comparaciones entre valores enviados y recibidos. La vista Test Data View muestra el contenido de un mensaje en particular y pueden verse las propiedades de Test Case incluyendo su veredicto.

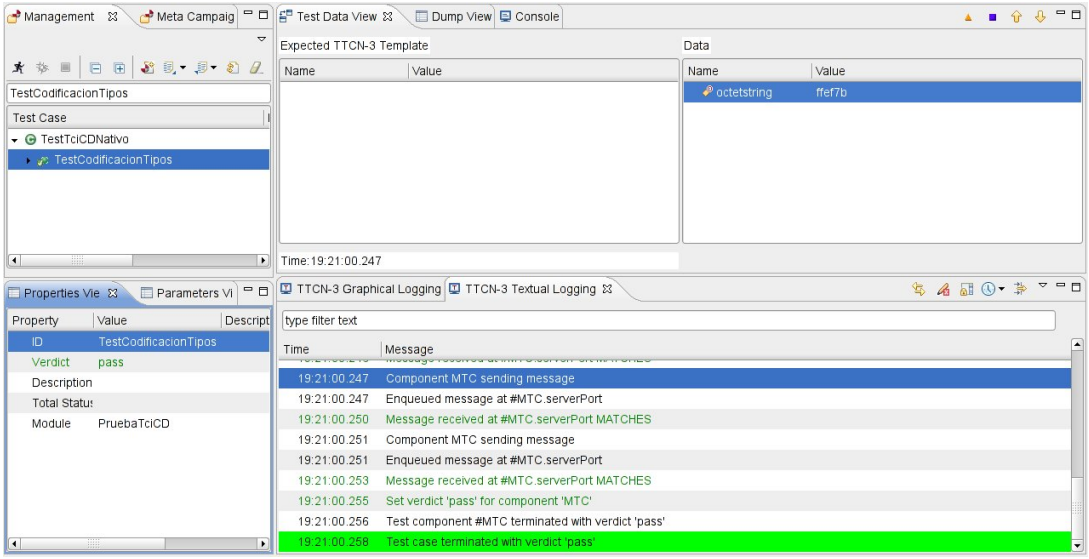


Figura 6.4: PruebaTciCD.ttcn3 (TTman)

La salida de consola y el archivo de Logging para la ejecución pueden verse en el Apéndice D. .

6.1.3. Integración del Adapter Nativo

Luego de que se probó y validó la integración inicial con el T3DevKit la siguiente etapa es probar la implementación del Adapter, completando de esta forma la integración para las interfaces TRI/SA y TRI/SA con el T3devKit.

Una prueba básica se implementa en la clase `TestTciAdapter.java`, la cual lee la configuración para un Codec y carga de la biblioteca dinámica. Desde cualquier clase principal Java puede ser instanciada esta clase para poder verificar la recuperación del Codec correcto y que se puede cargar la biblioteca dinámica sin problemas.

Para ajustar y validar la integración para la interfaces TRI se utilizaron una batería de Test Cases incorporados por el T3DevKit.

El primer módulo TTCN-3 utilizado es el `DNSTester.ttcn3`, el cual incorpora las sentencias para realizar las pruebas sobre el protocolo de DNS; luego se utilizó el módulo `DNSTester2.ttcn3` el cual también prueba el protocolo de DNS pero con estructuras más complejas y mejoradas teniendo así un caso de test más realista para el protocolo. En tercer lugar se probó la utilización de CoDec mixtos, ejecutando el módulo `MixedCoDec.ttcn3`, aquí se definen tipos a ser codificador por el CoDec generado por el T3DevKit y otros tipos que serán codificados por un CoDec diferente.

Lo importante en esta etapa es probar la integración de la entidades participantes para la implementación TRI para ambas API y la ejecución de los módulos TTCN-3

mencionados, estas pruebas ayudaron en el ajuste de la implementación de las interfaces.

Desde el TTCN-3 Graphical Logging se exportó la Figura 6.5, donde se muestra la secuencia de envíos y recepciones en la ejecución de la prueba DNSTester. También puede verse como fue utilizado el temporizador y el momento de la comparación entre el valor recepcionado y el valor esperado, lo que determina el veredicto `pass` en la prueba.

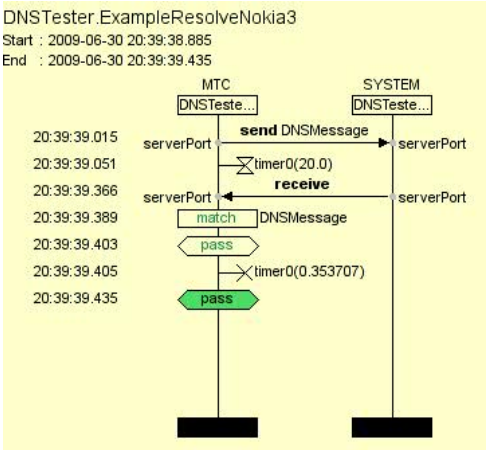


Figura 6.5: DNSTester.ttcn3 (Graphical Logging)

En la Figura 6.4 se muestra la vista del TTCN-3 Textual Logging para la ejecución del test DNSTester. En la misma se detalla el seguimiento de los envíos, recepciones y comparaciones entre valores enviados y recibidos y la vista Test Data View muestra el contenido del DNSMessage y pueden verse las propiedades de Test Case incluyendo su veredicto.

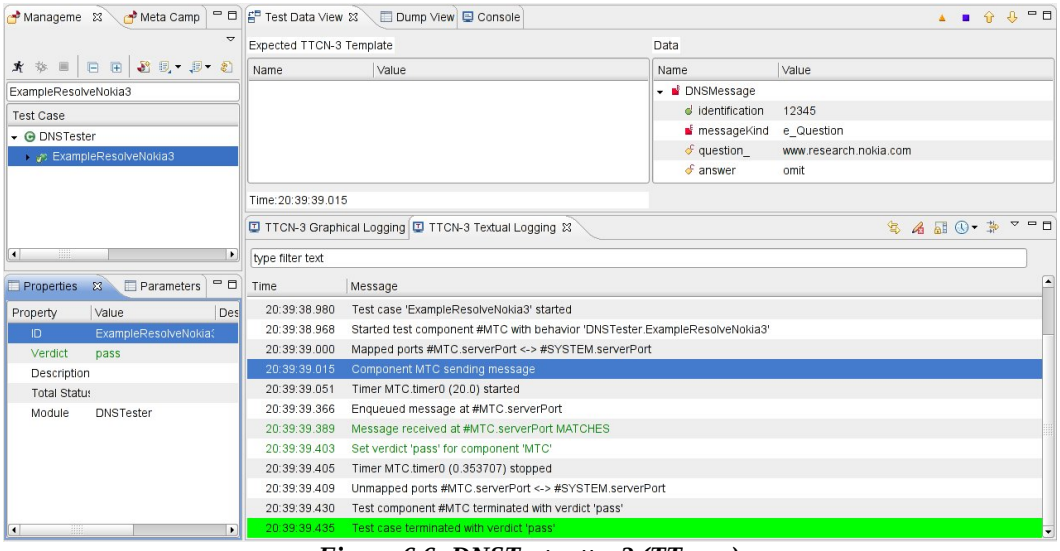


Figura 6.6: DNSTester.ttcn3 (TTman)

La salida de consola y el archivo de Logging para la ejecución pueden verse en el Apéndice E. .

Con ejecuciones de los test DNSTester2 y MixedCoDec se obtiene resultados gráficos análogos los obtenidos para la prueba DNSTester, por lo que las figuras 6.5 y 6.6 son tomadas como referencia la hora de analizar los resultados gráficamente.

El último Test Case ejecutado es el `RIPng`, éste tiene la particularidad de que se utiliza importación de módulos e invocación a funciones externas. Este test fue el único que no pudo ser completado con éxito ya que la codificación de parámetros para funciones externas implementado por TTwb no es compatible con el manejo que realiza el T3DevKit. Por parte del T3DevKit se emplean las funciones de codificación/de-codificación estándares para parámetros y valores de retorno en las funciones externas, mientras que el TTwb utiliza una clase CoDec particular (`com.testingtech.ttcn.tci.codec.base.ParameterCodec`) para realizar estas tareas.

Como se puede ver en la Figura 6.8, se produce un error en la clase `ParameterCodec`, al intentare de-codificar un valor que para el T3DevKit esta representado como una cadena de caracteres, mientras que el TTwb espera un objeto entero, mientras que en la Figura 6.7 se denota que, no se alcanza a realizar el envío ya que el error se produce antes del mismo, lo que determina el veredicto `error` en la prueba.

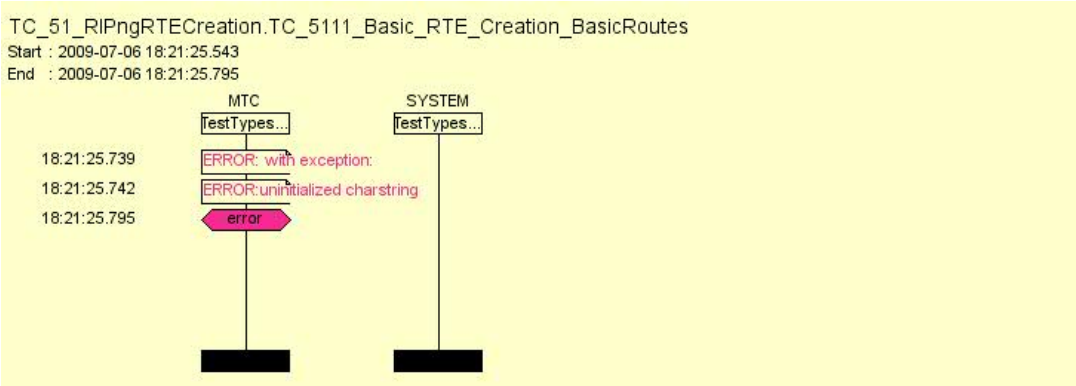


Figura 6.7: RIPng (Graphical Logging)

El TTCN-3 Textual Logging para la ejecución del test RIPng se muestra en la Figura 6.8, se puede ver que solamente se alcanzan a realizar las correspondencia de los puertos y luego se muestra la traza del error Java producido.

.

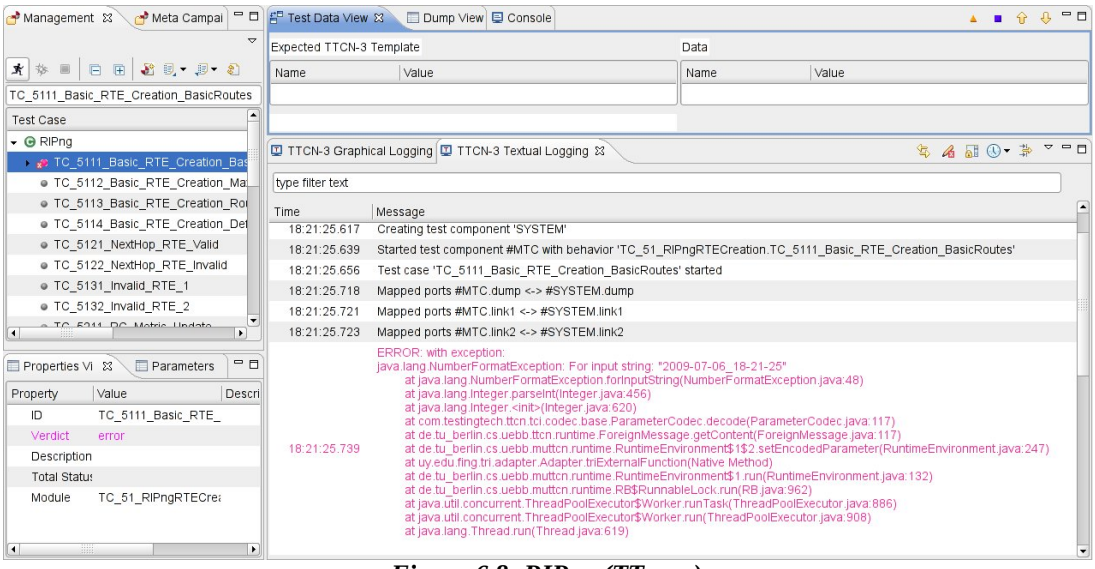


Figura 6.8: RIPng (TTman)

La salida de consola y el archivo de Logging para la ejecución pueden verse en el Apéndice F.

6.2. Resultados obtenidos

El testear primero el manejo de TCI Values fue muy productivo ya que se validó la factibilidad de la interacción entre las dos plataformas y se obtuvo una primera versión del conector. Por otra parte esto obligó a realizar un análisis detallado de las estructuras Java y C/C++ que representan a los tipos TTCN-3 y como realizar las conversiones de una a la otra.

También fue importante a la hora de analizar la arquitectura, diseño e implementación del TTwb, dado que en esta etapa de pruebas se estudió como configurar las clases Codec y Adapter, las cuales son utilizadas por el T3RTS.

La utilización del Codec con implementación nativa y la ejecución de las pruebas sobre el mismo, llevaron a la validación de la integración con las operaciones de codificación y decodificación implementadas por el T3DevKit. Analizando los resultados, después de realizar los ajustes necesarios fueron codificados y decodificados los tipos TTCN-3 básicos y también los definidos por el usuario.

En esta primera etapa de interacción con el T3DevKit se completaron las invocaciones Java → C y C → Java para las interfaces TCI/CD y TCI Values, el éxito de estas pruebas habilitó a continuar con la integración de las interfaces TRI/SA y TRI/PA.

Finalmente al contar con la integración total entre los módulos Java y C/C++ y disponer de un conector implementado en forma total y completamente funcional, se pudieron ejecutar los Test Case: DNSTester, DNSTester2 y MixedCoDec con resultados exitosos. Esto cubre de manera satisfactoria las pruebas sobre un conjunto de

funcionalidades fundamentales para la generación de CoDecs y su utilización desde el TTwb.

Con respecto al Test Case RIPng, cabe mencionar que fue el único test que no tuvo un resultado exitoso. Aquí se realizan invocaciones a funciones externas y se detecto que el resultado toma un veredicto `error`, ya que se utiliza una codificación no estandarizada para los parámetros y valores de retorno dentro del TTwb. Se trato de agregar algún tipo funcionalidad no genérica, por ejemplo: una solución *hard-coded* para manejar algunas representaciones particulares para los parámetros dentro del conector e intentar que el Test Case funcionara correctamente, pero esto no fue posible, dado que, no hay manera de determinar como convertir los valores recibidos por el T3DevKit a lo esperado por el entorno TTwb, ya que aquí se utilizan clases que no tiene código abierto.

Teniendo en cuenta los resultados de las pruebas, se pudo comprobar y validar empíricamente la interoperabilidad entre plataformas, verificando uno de los principales objetivos trazados al comienzo del proyecto.

7. Otros trabajos y reconocimientos vinculados

En esta sección se mencionan los trabajos y participaciones en otros proyectos, cursos y actividades relacionadas directa o indirectamente con este trabajo y la tecnología TTCN-3.

La dedicación a este tipo de actividades fue muy importante, aportando experiencia, practica y conocimientos que fueron volcados en la resolución de esta tesis; aunque por otra parte, influyó en que la entrega final del trabajo se extendiera en el tiempo.

7.1. Colaboración en el trabajo de extensión de picoTTCN-3

Se colaboró con Ricardo Rezzano en el trabajo de extensión de picoTTCN-3 de Go4IT para contar con una API dual [23]. La importancia de este trabajo, es contar con una plataforma dual, donde se posible aprovechar los puntos fuertes de cada uno de los lenguajes simultáneamente desde TTCN-3. Generalmente, C/C++ se asocia con trabajo a más bajo nivel y es más adecuado a la hora de implementar rutinas referentes a protocolos de comunicación. Por otra parte, Java brinda un marco de trabajo orientado a objetos, además de contar con un amplio conjunto de bibliotecas a la hora de trabajar con problemas de índole general.

Dada la experiencia adquirida en la tecnología JNI, se brindo soporte y apoyo en el diseño e implementación de los módulos de picoTTCN-3 que la requerían al momento de realizar la integración con el T3RTS.

Fue necesario realizar un aporte en base a los conocimientos sobre la interfaces ETSI y las correspondencias de los tipos de datos abstractos a los lenguajes de plataforma, realizando así diferentes tareas dentro de los objetivos trazados en el proyecto Go4IT.

Se ayudo a generar la documentación referente a la inclusión de la tecnología JNI y el análisis de este trabajo.

7.2. Student Grant T3UC-2008

Debido al avance en este trabajo y los conocimientos adquiridos se obtuvo el "Student Grant T3UC-2008". La T3UC (TTCN-3 User Conference – Conferencia de Usuarios de TTCN-3) se realizó en el año 2008 en la ciudad de Madrid, España.

Gracias al apoyo financiero, fue posible el haber asistido a la conferencia y significó una experiencia muy significativa, pudiendo conocer las distintas aplicaciones del lenguaje, la nuevas tendencias e investigaciones sobre la tecnología, teniendo la posibilidad de compartir con las personas referentes en la misma.

La carta presentada al comité evaluador, en base al cual se nos otorgó la beca estudiantil, se incluye en el Apéndice G. .

7.3. Curso de Postgrado "Testing de Software"

Se participó en el dictado del Curso de Postgrado "Testing de Software" en el marco del CPAP, octubre 2008. Teniendo a cargo la practica sobre el lenguaje TTCN-3.

En el contexto del curso fue necesario aplicar la experiencia acumulada a lo largo del trabajo con TTCN-3 para poder mostrar particularmente el compilador TTwb e introducir a los alumnos en las generalidades de la tecnología incluida en el lenguaje y su arquitectura.

Se implementó un ejemplo simple, donde se exponía el manejo de tipos básicos TTCN-3 y tipos definidos por el usuario, en su codificación y de-codificación, poniendo énfasis en la clara definición de las interfaces que facilitan la interacción entre las entidades participantes en la ejecución de un Test Case. Se utilizaron las herramientas para el análisis gráfico de las ejecuciones, logrando indicar como se realiza la secuencia de llamadas entre las entidades y pudiendo visualizar concretamente el contenido de los mensajes involucrados en la prueba.

7.4. INRIA International Internship

En base a los conocimientos adquiridos sobre TTCN-3 y el T3DevKit, se participó en la selección para el programa INRIA International Internship del año 2008, resultando aceptado en el equipo Dionysos del centro de investigación Rennes-Bretagne Atlantique, durante un semestre.

La pasantía fue titulada: "T3DevKit and mu-TTCN-3 unified parsing". Este proyecto surge, en base a que, IRISA desarrolla y mantiene el TTCN-3 CoDec Generator, herramienta que automatiza a las tareas de implementación para TTCN-3 TCI/CD, y también, coopera en el proyecto internacional Go4IT para el desarrollo de un compilador TTCN-3 de código abierto y libre. Ambas herramientas están implementadas en C/C++ y comparten la misma tecnología de base, utilizando parsers del lenguaje TTCN-3 fundados en Bison/Flex.

Como tarea inicial, se analizaron las funcionalidades de ambos parsers y su estado actual, reconociendo las limitaciones de cada uno. Entonces, surge la idea de diseñar e

implementar una librería de utilidades que brinde los servicios necesarios a ambos analizadores para complementar la funcionalidades existentes.

Luego de definidos los requerimientos, se realizó el desarrollo de un módulo C++, con un fuerte enfoque en las interfaces que la biblioteca define para la interacción con los parsers y el otro objetivo importante fue entregar una documentación completa y adecuada sobre el trabajo, a nivel de código fuente y presentación del informe final

Otros trabajos y reconocimientos vinculados

8. Conclusiones

En esta sección se presentan los resultados del proyecto, tomando en cuenta las tareas realizadas para generar el conector, las practicas incluidas en los trabajos relacionados y la utilización de las herramientas integradas.

8.1. Sumario

Las herramientas que fueron utilizadas para desarrollar la solución propuesta tenían algunas limitaciones conocidas a priori, las mismas trataron de ser solucionadas durante el proyecto. El T3DevKit no es completamente independiente de la implementación de las interfaces que brinde el compilador TTCN-3 utilizado, esto se comprobó en la practica y hubo que realizar ciertas adaptaciones al mismo para poder utilizar el compilador TTwb. El TTwb implementa las interfaces ETSI estandarizadas y además utiliza ciertas interfaces propietarias; se encontraron como era previsto algunas limitaciones a la hora de la interoperación con el T3DevKit, específicamente las invocaciones a funciones externas no pudieron ser ejecutadas porque el compilador TTwb maneja una codificación de parámetros y valores de retorno no estándar.

El lenguaje TTCN-3 contiene muy buenas características para la especificación simple, abstracta y sin ambigüedad de pruebas y su comportamiento esperado, definiendo claramente su resultado final o veredicto. Este nivel de abstracción ayuda a la portabilidad y re-utilización de definiciones para casos de pruebas.

La curva de aprendizaje con respecto a la tecnología detrás del lenguaje TTCN-3 está determinada en gran medida por la implementación o construcción de las entidades que interactúan a la hora de la ejecución de un Test Case, esto es la generación de los CoDecs y la implementación de las interfaces de la TRI. A lo largo de este trabajo se estudiaron diferentes variantes de como generar estas entidades automáticamente, pudiendo evaluar las diferentes opciones con respecto a la facilidad y simplicidad a la hora de su incorporación en la ejecución de las pruebas.

En el dictado del curso Testing de Software se debieron generar manualmente todas las entidades en lenguaje Java para poder mostrar la ejecución del Test Case en la practica sobre TTCN-3, no se utilizó código generado y se pudo comprobar que esta tarea no es trivial ni pequeña, dado que el implementador debe tener buenos conocimientos sobre el lenguaje Java, las interfaces del estándar y además conocer ciertos requerimientos sobre el SUT.

Entre las necesidades que llevaron a crear el T3DevKit está que el desarrollo y especialmente el mantenimiento de un CoDec es una tarea costosa, esto también aplica para la construcción de los System y Platform Adapter. Las dificultades encontradas el lenguaje C/C++ son similares a las encontradas en la práctica con lenguaje Java al respecto del grado de dificultad que tienen estas tareas.

Esto justifica la discusión sobre las ventajas de utilizar el conector JNI para reusar la implementación del T3DevKit, el cual se vale de la generación de CoDecs complementada con la inclusión de pequeñas partes de código C/C++ (Codets, únicas porciones de código a implementar por el usuario en base a ciertas características sobre los tipos TTCN-3 definidos).

8.2. Conclusiones finales

TTCN-3, al estar este especificado bajo un estándar, se considera que puede ser aplicado de forma genérica en el testing de cualquier tipo de sistema, equipo y/o disciplina; pero, en estos momentos, su utilización se sigue dando, casi exclusivamente en el área de las telecomunicaciones. Se ha comprobado la efectividad del lenguaje para el testeo en esta área, logrando altos niveles de automatización de las pruebas en la misma, especialmente en los procesos de conformance para IPv6.

Revisando otras áreas de aplicación, vemos que el testeo de un servicio web podría ser desarrollado eficientemente con TTCN-3, ya que publica interfaces fuertemente tipadas y no incorpora interfaz de usuario, por lo que no sería muy costoso escribir los módulos TTCN-3 que prueben la interacción con el servicio. En otro tipo de sistemas vemos bastante más dificultosa su aplicación; por ejemplo el testeo de paginas web o pruebas sobre sistemas con interfaz de usuario ya que no tiene abstracciones para manejar este tipo de entidades.

Evaluando el TTwb entre sus fortalezas podemos destacar que cuenta con un excelente mecanismo para el seguimiento de las ejecuciones de los tests y permite su re-ejecución de manera sencilla. Su ambiente de desarrollo cuenta con utilidades que facilitan el uso de TTCN-3. Además cuenta con una cobertura completa de las interfaces definidas en el estándar facilitando su integración. Encontramos algunas dificultades para la integración debido a la existencia de algunas extensiones no estándares.

La generación del conjunto de entidades necesarias para llevar a cabo la interacción con el SUT, pudiendo ejecutar un caso de prueba escrito en lenguaje TTCN-3, no es una tarea fácil y la curva de aprendizaje sobre estas prácticas es realmente alta, lo que consideramos va en detrimento de la promoción y el uso de TTCN-3. Esta razón motivó la creación del T3DevKit.

Respecto al T3DevKit se puede concluir que la generación automática de CoDecs a partir del código fuente TTCN-3, facilita el uso del lenguaje y es un gran avance para disminuir la dificultades a la hora de implementar las entidades necesarias para la creación de un testcase. El usuario no necesita conocer ni modificar el código generado, haciéndolo más fácil de mantener.

Estudiando las alternativas existentes para implementar una solución equivalente al T3DevKit para Java la posibilidad de re-codificarlo en Java fue descartada, surgiendo como la mejor solución la construcción del conector JNI.

Consideramos que la implementación del conector brinda mayores beneficios a la hora de contar con un generador automático de CoDecs en Java ya que esta solución re-utiliza las funcionalidades existentes del T3DevKit y minimiza el esfuerzo de desarrollo. Por otra parte si se tuviera una versión Java del T3DevKit las tareas de mantenimiento se incrementarían y sería más complicadas, teniendo que impactar en dos módulos separados cada vez que se requirieran cambios y ajustes por cualquier motivo.

Independientemente del lenguaje con que esté implementado el compilador TTCN-3 (actualmente solo existen implementados en Java o C/C++), a partir de una especificación para ATS escrita en TTCN-3 junto a uno o varios Codets, se obtienen las entidades necesarias para ejecutar un Test Case. Como resultado principal este trabajo podemos afirmar que es tecnológicamente posible la incorporación de un conector JNI para integrar el T3DevKit a un compilador Java, quedando operativas una versión de TTwb y T3DevKit que permitieron crear y ejecutar los ejemplos incluidos en el T3DevKit desde un TE en Java.

8.3. Trabajo futuro

Al momento de retomar y complementar este trabajo se pueden tener en cuenta algunas funcionalidades que no han sido completadas, ya que no son necesarias tomando en cuenta el estado actual de las herramientas integradas. Especialmente, se debe realizar un seguimiento en la evolución del T3DevKit, revisando sus próximas versiones.

En lo que respecta al conector JNI, sería interesante completar la implementación de los métodos stub. Las correspondencias para los TCI Values `Address` y `UniversalCharstring` deberían ser completadas en futuras versiones del conector, así como también, las funciones TRI para el manejo de invocaciones remotas broadcast y multicast. Fue verificado, que los métodos para invocaciones remotas, no son soportados actualmente por el T3DevKit, lo que implica una dependencia con esta herramienta a la hora de poder probar estas funcionalidades.

Estudiar adaptaciones para realizar invocaciones a funciones externas, es una tarea no trivial, pero debería tomarse en cuenta a futuro. Es esto implicará que el código C dentro del conector que realiza las correspondencias sea más complejo y tengan más responsabilidades, con el objetivo de llevar a cabo las transformaciones necesarias para cumplir con la representación Java para codificación de los parámetros y valores de retorno en funciones externas.

Se conoce que, Go4IT ha implementado las interfaces ETSI en C++ con orientación a objetos. Esto no ha sido utilizado en esta versión del conector y sería interesante, hacer que el conector manejara ambas representaciones para los TCI Values (estructuras de datos C y clases C++) e interfaces TCI y TRI. En otras palabras, se debería complementar la implementación del conector con las transformaciones de objetos Java a clases C++ y la generación de los mismos desde clases C++.

Conclusiones

Actualmente, los módulos C que componen el conector, brindan cada uno su archivo makefile para ser compilados. Una posible mejora sería, automatizar las tareas de compilación con el uso de Automake, logrando la generación automática de los makefiles y permitiendo configurar la compilación de los módulos de manera centralizada.

9. Glosario

ANSI:	American National Standards Institute.
API:	Application Programming Interface.
ASCII:	American Standard Code for Information Interchange.
ATS:	Abstract Test Suite.
BMP:	Basic Multilingual Plane.
CPAP:	Centro de Postgrados y Actualización Profesional.
CD:	Coding/Decoding.
CDGen:	CoDec Generator.
CL:	Core Language.
CH:	Component Handler.
CoDec:	Encoder/Decoder.
Codet:	Módulo que complementa a un CoDec en el marco de trabajo del T3DevKit.
Danet:	Devoteam Danet (TTCN-3 Toolbox).
DNS:	Domain Name System.
Eclipse:	Plataforma de programación, usada para crear entornos integrados de desarrollo.
EDS:	(Internal) Encoding/Decoding System.
ETS:	Executable Test Suite.
ETSI:	European Telecommunications Standards Institute.
GFT:	Graphical Presentation Format.
GIF:	Graphics Interchange Format..
Go4IT:	Advanced Tools and Services for Ipv6 Testing.
GUI:	Graphical User Interface.
IDE:	Integrated Development Environment.

IDL:	Interface Definition Language.
IPv6:	Internet Protocol version 6.
IRISA:	Institut de recherche en informatique et systèmes aléatoires.
IUT:	Implementation Under Test.
IXIT:	Implementation eXtra Information for Testing.
JNI:	Java Native Interface.
JNIEnv:	JNI Environment.
JVM:	Java Virtual Machine.
Logging:	Registro.
MSC:	Message Sequence Chart.
MTC:	Main Test Component.
OMG:	Object Management Group.
OSS:	Open Source Software.
PA:	Platform Adaptor.
Pass:	Valor para el veredicto en caso de que se pase el Test Case.
picoTTCN-3:	TTCN-3 compiler OSS.
Port:	Interfaz abstracta para facilitar la comunicación entre componentes del test.
PTC:	Parallel Test Component.
RIPng:	Routing Information Protocol next generation.
RB:	Runtime Behavior.
SA:	SUT Adaptor.
SBP:	Sistema bajo pruebas.
SUT:	System Under Test.
TCI:	TTCN-3 Control Interfaces.
TE:	TTCN-3 Executable.
Telelogic:	Telelogic AB (TTCN Suite).
TFT:	Tabular Presentation Format.
TID:	Timer Identification.
TL:	Test Logging.
TM:	Test Management.
TMC:	Test Management and Control.
TRI:	TTCN-3 Runtime Interfaces.

TSI:	Test System Interface.
TT:	TestingTech.
TTCN:	Testing and Test Control Notation.
TTCN-3:	Testing and Test Control Notation Version 3.
TTman:	TTCN-3 Execution Management.
TTwb:	TTworkbench.
T3CDGen:	TTCN-3 CoDec Generator.
T3DevKit:	TTCN-3 Development Kit.
T3DevLib:	TTCN-3 Development Library.
T3RTS:	TTCN-3 RunTime System.
T3UC:	TTCN-3 User Conference.
Wrapper:	Envoltorio, que a lo largo de este trabajo es sinónimo de Conector.

Glosario

Referencias

Bibliografía

1: ETSI, TTCN-3 Standard Suite, último acceso 2009-07-03, http://www.ttcn-3.org/StandardSuite.htm
2: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3;Part 1: TTCN-3 Core Language - ETSI ES 201 873-1 V4.1.1 (2009-06), 2009
3: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics - ETSI ES 201 873-4 V4.1.1 (2009-06), 2009
4: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI) - ETSI ES 201 873-5 V4.1.1 (2009-06), 2009
5: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3;Part 6: TTCN-3 Control Interface (TCI) - ETSI ES 201 873-6 V4.1.1 (2009-06), 2009
6: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3;Part 7: Using ASN.1 with TTCN-3 - ETSI ES 201 873-7 V4.1.1 (2009-06), 2009
7: ETS, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3;Part 9: Using XML schema with TTCN-3 - ETSI ES 201 873-9 V4.1.1 (2009-06), 2009
8: Testing Technologies A.G., TTworkbench - The Reliable Test Automation Platform, último acceso 2009-06-22, http://www.testingtech.com/products/ttworkbench.php
9: INRIA, TTCN-3 Development Kit, último acceso 2009-06-22, http://gforge.inria.fr/projects/t3devkit/
10: Sun Microsystems, Inc. , Java Native Interface, 2003 (último acceso 2009-06-24),
11: Go4IT, Advanced Tools and Services for IPv6 Testing, último acceso 2009-07-03, http://www.go4-it.org/
12: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3;Part 6: TTCN-3 Control Interface (TCI) - ETSI ES 201 873-6 V3.2.1 (2007-02), 2007

Referencias

13: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3;Part 5: TTCN-3 Runtime Interface (TRI) - ETSI ES 201 873-5 V3.2.1 (2007-02), 2007
14: ETSI, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language - ES 201 873-1 V3.2.1 (2007-02), 2007
15: Colin Willcock, Thomas Deiß, Stephan Tobies,Stefan Keil, Federico Engler and Stephan Schulz, An Introduction to TTCN-3, 2005, Jhon Wiley & Sons, 13 978-0-470-01224-6
16: Testing Technologies A.G., Home page, último acceso 2009-06-22, http://www.testingtech.com/
17: Free Software Foundation, Inc., Automake - GNU Project - Free Software Foundation (FSF), último acceso 2009-07-03, http://www.gnu.org/software/automake/
18: Sheng Liang, Java Native Interface: Programmer's Guide and Specification, 1999, Addison-Wesley, 9780201325775
19: Fernando López Hernández, JNI: Java Native Interface, 2007, http://macprogramadores.org/tutoriales/tutoriales/jni.pdf
20: H. M. Deitel - P. J. Deitel, Como Programar en C/C++, 1998, Prentice Hall, 9688804711
21: Con Clase, C++ con Clase, 2004 (último acceso 2009-06-24), http://c.conclase.net/
22: Sun Microsystems, Inc. , The Java Tutorials, 2009 (último acceso 2009-06-24), http://java.sun.com/docs/books/tutorial/
23: R. Rezzano, A. Sabiguero, C. Viho, TTCN-3 Tools Interoperability Between Java and C/C++ Platforms, 2008, http://www.fing.edu.uy/~asabigue/publi/rrezzanot3uc2008paper.pdf

A. Apéndice

Documentación sobre la instalación del compilador TTwb.

Este apéndice incluye el contenido de un documento entregable intermedio, solicitado por el supervisor del proyecto, a los efectos de contar con instrucciones claras de como instalar y configurar el entorno de desarrollo TTCN-3 brindado por el Testing Tech.

1 Requerimientos

Para la utilización de la herramienta se debe contar con el hardware y software que se describirá en los siguientes apartados.

1.1 Sistema Operativo

- Windows XP, 2000, 98, ME.
- Linux Red Hat ≥ v7.1 (x86/GTK).
- Linux SuSE ≥ v9.1 (x86/GTK).

1.2 Memoria RAM

- 512 MB o más (1 GB recomendado).

1.3 Plataforma Java 2 (JRE)

- Al menos, contar con la versión J2SE RE 5 (1.5.x). Descargas disponibles desde: http://java.sun.com o http://www.ibm.com/developerworks/java/jdk/

2 Descarga del producto

Debe ser realizada desde el portal de Testing Technologies: http://www.testingtech.de/support/downloads.php.

Nota: la descarga refiere al producto destinado para uso académico (básico y con fecha de espiración para la licencia), por lo que para se debe tener previamente un **Reference ID** (número de autentificación) para realizar la misma.

2.1 Selección

De la lista de productos disponibles, se debe seleccionar la última versión del TTworkbench (como se mencionó antes, trabajaremos con el producto Basic) para la plataforma que se disponga.

2.2 Autentificación

Ingresar el **Reference ID** destinado a la descarga.

2.3 Descarga

Descargar y guardar el archivo instalador en el directorio de destino deseado.

2.4 Localización del archivo con información sobre la licencia

Guardar el archivo *license.dat* (este archivo sera utilizado en la instalación del producto).

3 Instalación

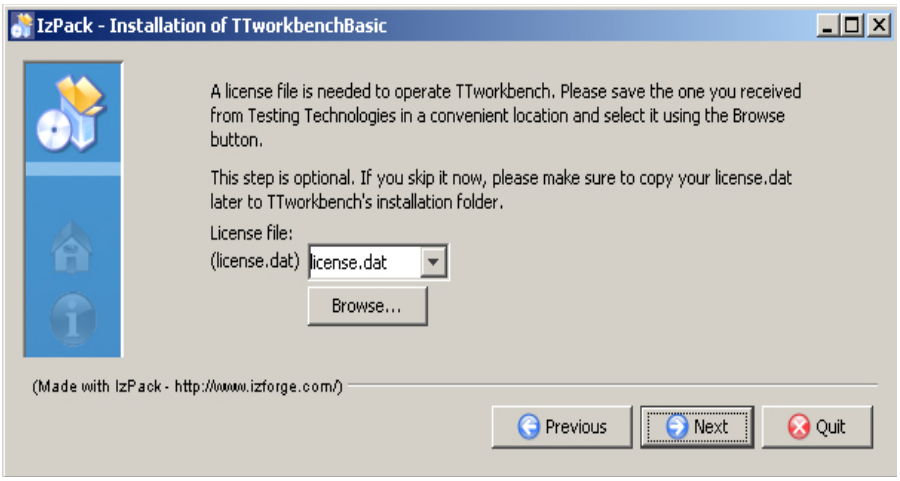
Las descripción de la instalación y todos los pasos subsiguientes están enfocados al manejo sobre la plataformas Windows y Linux, en particular basada en la experiencia adquirida en el trabajo sobre Windows XP Profesional y OpenSuse 11.1.

3.1 Inicio de asistente de instalación

Ejecutar el asistente de instalación (doble clic al archivo *TTworkbenchBasic-v1.1.x-installer.exe* en Windows o 'java *-jar TtworkbenchBasic-linux32-v1.1.x-installer.jar*' desde línea de comandos en Linux).

3.2 Selección de archivo con la información de la licencia

Siguiendo la instrucciones del asistente se llegará a un diálogo como el siguiente:

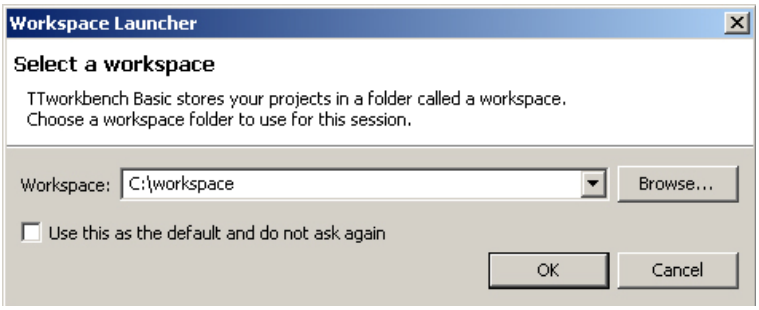


Luego, se debe localizar el archivo *license.dat*, previamente obtenido y se le debe indicar al asistente la ubicación del mismo (el TTworkbench requiere un archivo de licencia válido para su ejecución).

3.3 Ejecución de la herramienta

Para ejecutar el TTworbench se puede buscar su ejecutable en *Inicio->Programas->Testing Technologies->TTworkbenchBasic->TTworkbenchBasic* o vía el Acceso Directo creado en el Escritorio (TTworkbenchBasic).

Al ejecutarse el programa aparecerá el siguiente diálogo:



Se puede seleccionar el espacio de trabajo por defecto o elegir uno a conveniencia en la ubicación que se indique.

4 Activación de la licencia

Para activar la licencia, el archivo **license.dat** debe ser adjuntado a un servidor brindado por el proveedor del producto. Es importante verificar que dentro del archivo de la licencia, aparezca la configuración correcta para la conexión; o sea que, en el contenido del mismo deben aparecer como datos del servidor, el nombre del equipo y la MAC correspondientes a la máquina en la cual se trabajará con el TTworkbench (por ejemplo: SERVER ces-test 00110960226d).

La versión mas reciente del servidor de licencias, se puede obtener directamente desde Macrovision en la siguiente dirección: http://www.globes.com/support/fnp_utilities_download.htm

4.1 Iniciar el Servidor Manejador de Licencia manualmente

Es recomendable (pero no excluyente), para tener fácil acceso a los archivos necesarios, generar un directorio “flexml” (por ejemplo: ubicado en la raíz del disco C en Windows o en '/usr/local' en Linux, como se mostrara a continuación) y copiar dentro del mismo, el archivo license.dat, los archivos ejecutables TTECH (“daemon brindado por el proveedor del producto”) y lmgrd (servidor que depende de la plataforma y su arquitectura).

Luego de preparado el directorio como se describió, se debe ejecutar el *lmgrd* como aplicación desde línea de comando con la siguiente sintaxis:

C:\flexlm> lmgrd -c license_file_list -L [+]debug_log_path (Para Windows) y

/usr/local/flexml> lmgrd -c license_file_list -L [+]debug_log_path (Para Linux).

Donde:

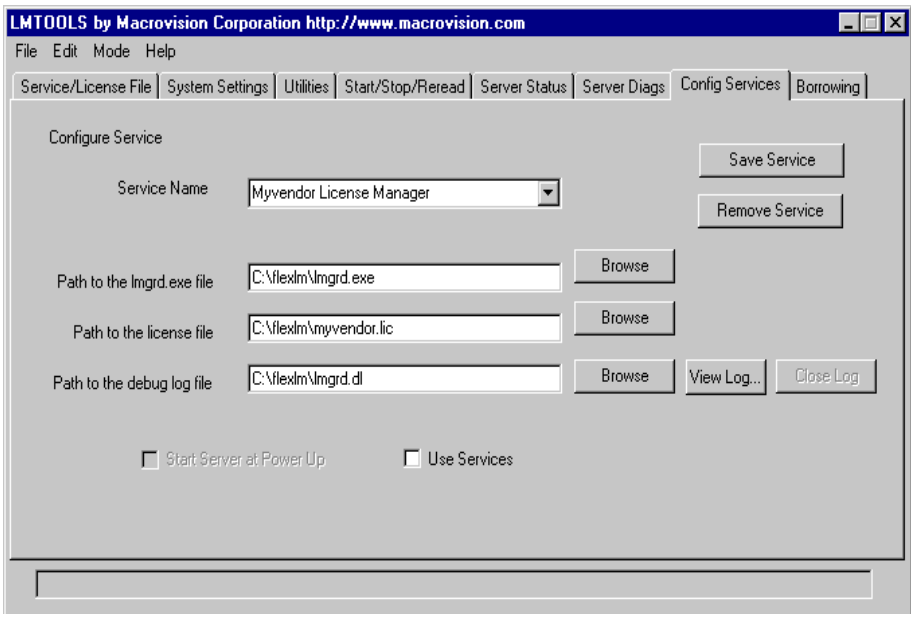
- *license_file_list* corresponde a una o mas de las siguientes opciones:
  - ruta completa a un solo archivo de licencia.
  - un directorio, donde todo los archivos con extensión "lic" dentro del mismo serán utilizados.
- *debug_log_path* corresponde la ruta completa al archivo que contendrá la traza servidor (archivo de log). Anteceder el *debug_log_path* con el carácter "+" hace que las entradas registradas se adjunten al final del archivo.

Nota: los espacios en las rutas requieren que las mismas sean rodeada por comillas dobles.

4.2 Configuración del Servidor como un servicio de Windows

Para realizar esta configuración se deben tener privilegios de Administrador y llevar a cabo los siguientes pasos.

1. Iniciar la herramienta LMTOOLS.
2. Seleccionar la opción *Configuration using Services*, luego ir a la lengüeta *Config Services*.
3. En el campo *Service Name*, ingresar el nombre que se quiera definir para el servicio.
4. En el campo *Path to the lmgrd.exe file*, ingresar o indicar la ruta al *lmgrd.exe* para este servicio.
5. En el campo *Path to the license file*, ingresar o indicar la ruta al archivo con la información de la licencia para este servicio.
6. En el campo *Path to the debug log file*, ingresar o indicar la ruta al archivo de log para este servicio. Anteceder el nombre del archivo con el carácter "+" hace que las entradas registradas se adjunten al final del archivo. La localización por defecto al archivo de log es la carpeta *c:\winnt\System32*. Para indicar una localización diferente, se deberá ingresar una ruta completa válida.



7. Para salvar la nueva configuración del servicio, se debe hacer clic en botón *Save Service*.

4.3 Iniciar el Servidor Manejador de Licencia desde el LMTOOLS (Windows)

Como se mostró en el punto anterior, se cuenta con una interfaz gráfica para el manejo de las herramientas del servidor de licencia llamada LMTOOLS.

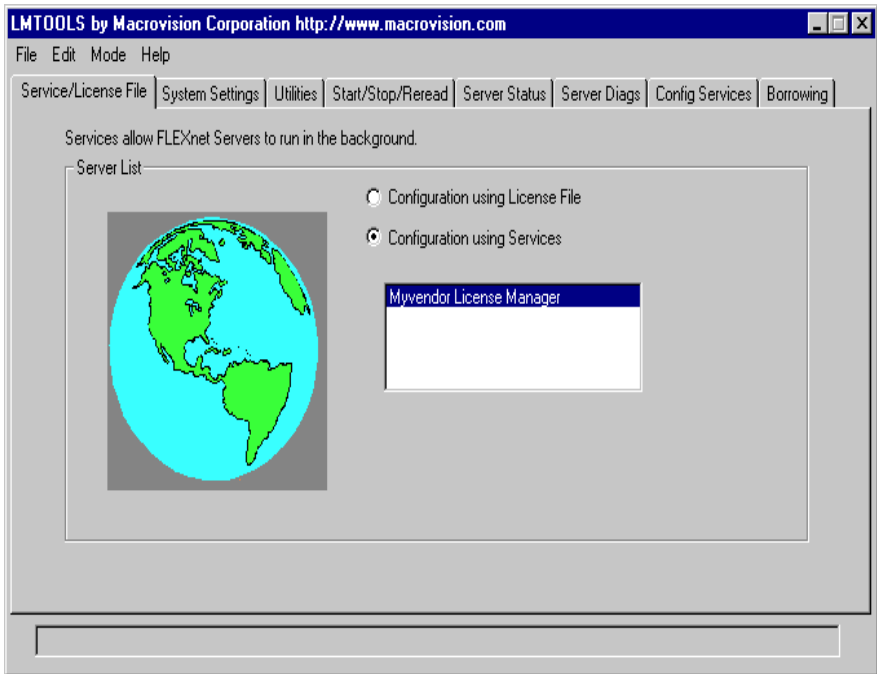
Dentro de las funcionalidades provistas se encuentran:

- iniciar, terminar y configurar el servicio FLEXnet para el servidor de licencia.
- obtener información del sistema, incluyendo los identificadores de host.
- obtener información sobre el estado del servidor.

Antes de poder ejecutar las operaciones de control sobre el *lmgrd* desde LMTOOLS, se deberá haber realizado antes la configuración descrita en el punto anterior.

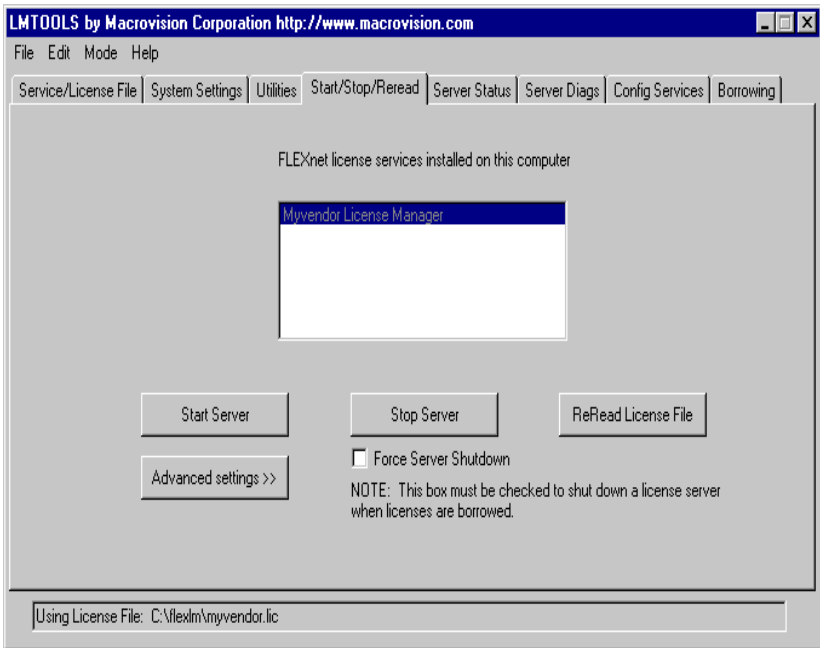
Una vez que el servicio fue configurado, el *lmgrd* puede ser iniciado como servicio desde el LMTOOLS como se muestra a continuación:

1. Ejecutar el LMTOOLS.



Al abrirse, aparecerá con la lengüeta *Service/License File* seleccionada.

2. Seleccionar la opción *Configuration using Services*.
3. Seleccionar el nombre del servicio desde la lista presentada.
4. Moverse a la lengüeta *Start/Stop/Reread*.

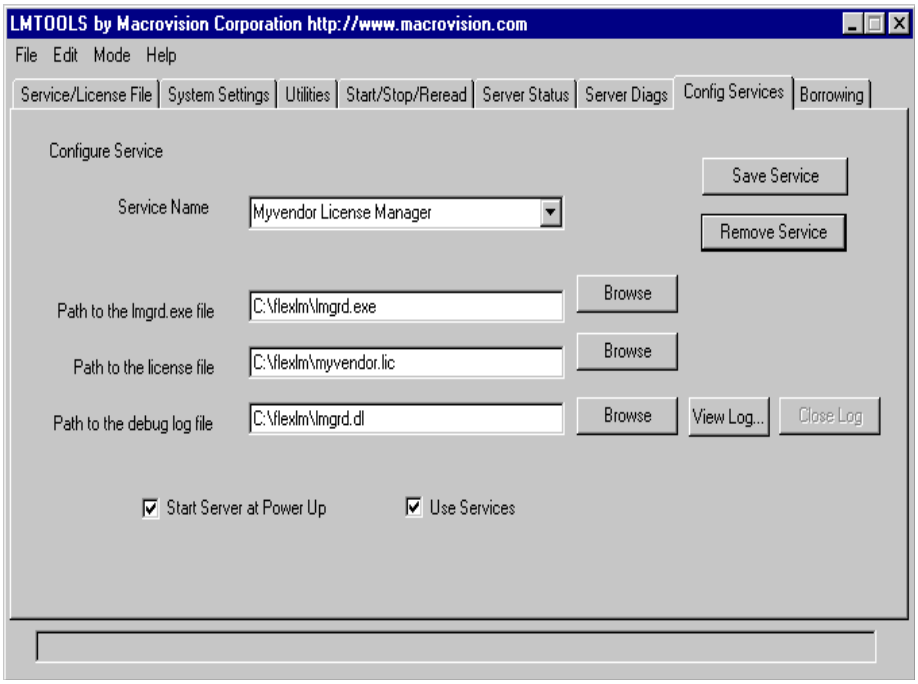


5. Iniciar el servicio haciendo clic en el botón *Start Server*. El servidor se iniciará y comenzará a escribir en el archivo de log especificado. Para detener el servidor se debe hacer clic en el botón *Stop Server* y en caso de que se quiera releer el archivo con la información de la licencia se debe hacer clic en el botón *ReRead License File*.

4.4 Inicio automático del servicio al iniciar Windows.

Para poder configurar el inicio automático del servidor cuando se inicia el Sistema Operativo, se debe tener configurado el servicio como se explicó anteriormente.

1. Con la la herramienta LMTOOLS iniciada y con el servicio a configurar seleccionado, moverse a la lengüeta *Config Services*.
2. Para hacer que el servicio sea manejado por Windows, se deberá chequear la opción *Use* Services (en cualquier otro caso, se convertirá en un servicio de licencia FLEXnet).
3. Para que el servicio se inicie al mismo tiempo que se inicie el Sistema Operativo, se deberá chequear la opción *Start Server at Power Up*.



A partir de ahora cada vez que el Sistema Operativo se reinicie el servicio manejador de la licencia se iniciará automáticamente como un servicio de Windows.

4.5 Inicio automático del servicio al iniciar Linux.

Para realizar esta configuración se deben tener privilegios de Administrador para editar algún boot script apropiado, por ejemplo: /etc/init.d/boot.local, e incluir una línea con una sentencia como la siguiente (dependiendo de la ubicación de los archivos ejecutables):

/usr/local/flexml/lmgrd -c /usr/local/TTworkbenchBasic/license.dat

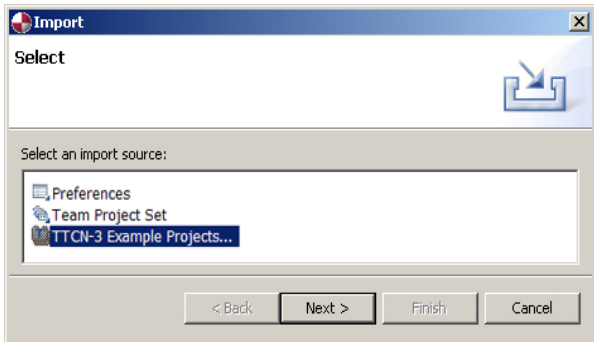
-L +/usr/local/flexml/lmgrd.log /usr/local/flexml/TTECH

5 Ejemplos

La herramienta incluye varios ejemplos que puede ser cargados como proyectos dentro del TTworkbench. Para esto se deben seguir lo que se describirá a continuación.

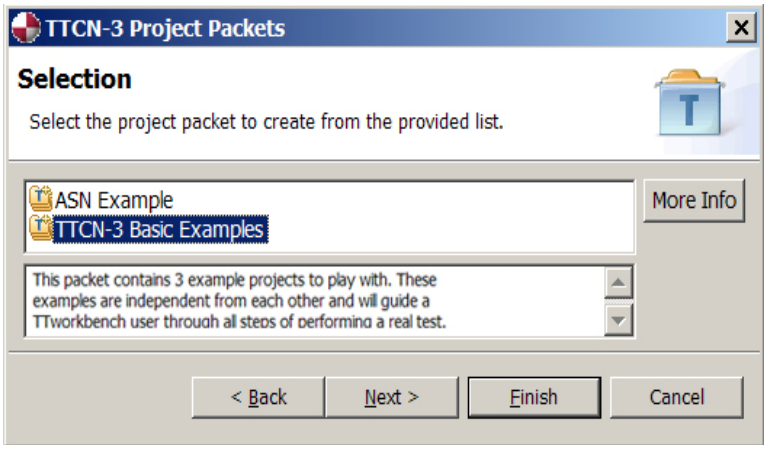
Importación de proyectos

Ingresar al Menú Principal en la opción *File*, seleccionar *Import...* y luego *TTCN-3 Example Projects...* para realizar el importación de los proyectos.



5.1 Selección de los ejemplos

Seleccionar *TTCN-3 Basic Examples* y seguir las instrucciones del diálogo que se muestra a continuación:



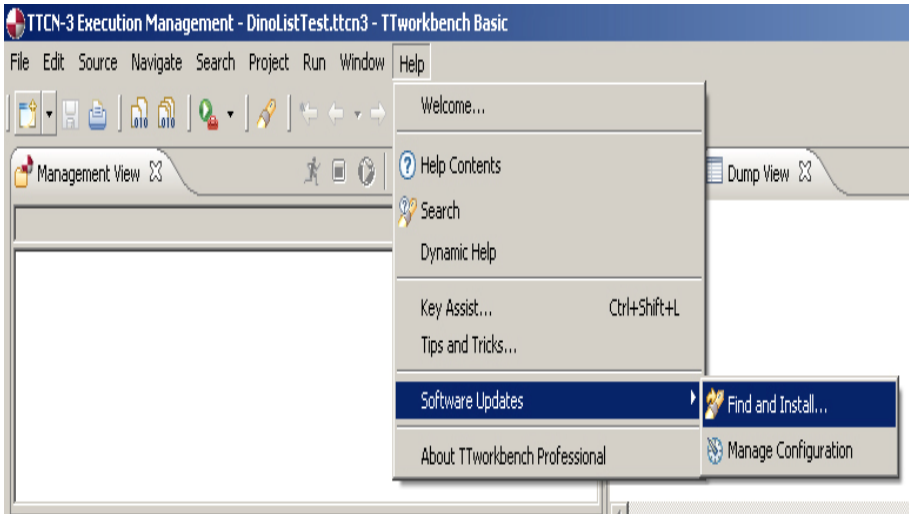
Después de esto, se puede encontrar los ejemplos que provee el TTworkbench en el la estructura de árbol dentro del Package Explorer.

6 Actualizaciones

Para obtener las actualizaciones de las últimas versiones de TTworkbench, se deben seguir los siguientes pasos para utilizar el mecanismo de actualización vía el sitio de actualizaciones:

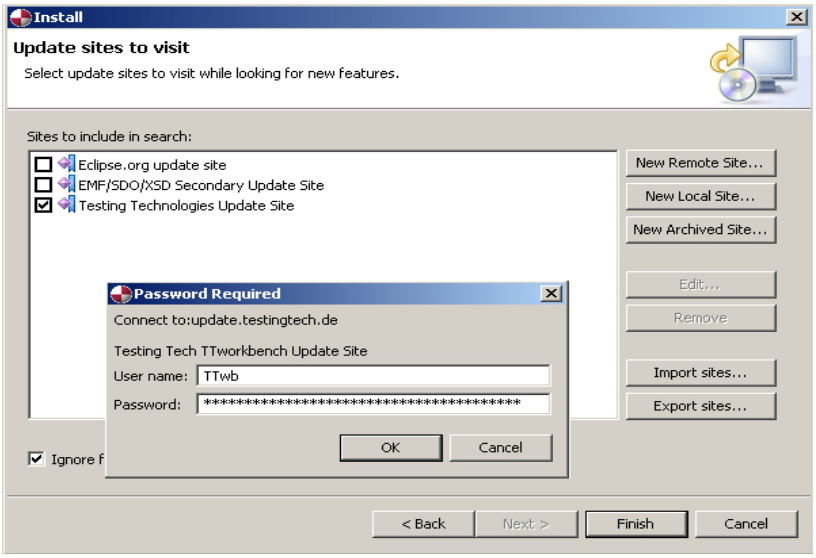
6.1 Ingreso al manejador de actualizaciones

Ingresar al Menú Principal en la opción *Help*, seleccionar *Software Updates–>Find and Install–>Search for new features to install…*



6.2 Selección del sitio y autentificación

Seleccionar *Testing Technologies Update Site*. Se debe realizar la autentificación, ingresando como nombre de usuario **TTwb** y como clave el **Reference ID**, como se muestra a continuación.



Se debe elegir TTworkbench, aceptar la EULA y comenzar la descarga.

Nota: como se muestra en la imagen, se pueden seleccionar otros sitios de actualización, que corresponden a funcionalidades del entorno Eclipse. Estos sitios no necesitan autentificación.

6.3 Efectivizar los cambios

Finalizada la descarga de los archivos de actualización, aparecerá la opción para reiniciar el TTworkbench, se debe aceptar para que los cambios tengan efecto.

7 Instalación del Cliente SVN y sincronización

Para obtener el cliente SVN que permita la sincronización con diferentes repositorios, se deben seguir los siguientes pasos:

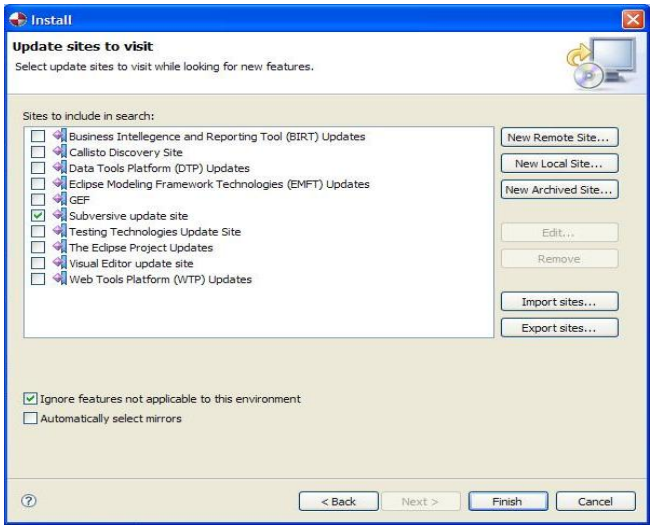
7.1 Configuración del sitio de descarga para el Cliente SVN

Se debe ingresar al Manejado de actualizaciones como se mostró en la sección *Actualizaciones*. Luego se le debe agregar al Manejador el sitio de descargas para el cliente SVN. Se debe ingresar el la opción *New Remote Site…* y allí completar los siguientes datos: en Name ingresar **Subversive** y en URL http://www.polarion.org/projects/subversive/download/1.1/update-site/.

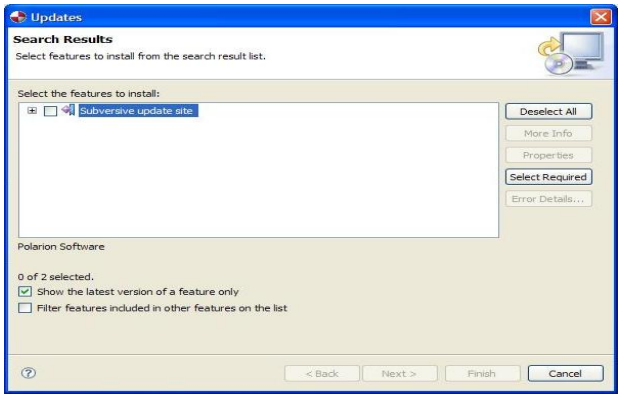
Después de ingresados los datos del sitio se debe confirmar haciendo clic en el botón *OK*.

7.2 Selección del sitio y descarga

Seleccionar *Subversive update site*, y hacer clic en el botón *Finish* para realizar la conexión con el sitio de descargas, como se muestra a continuación.



Luego deberá seleccionar las funcionalidades disponibles del Cliente SVN.



Se deberá proseguir, aceptar la EULA y comenzar la descarga.

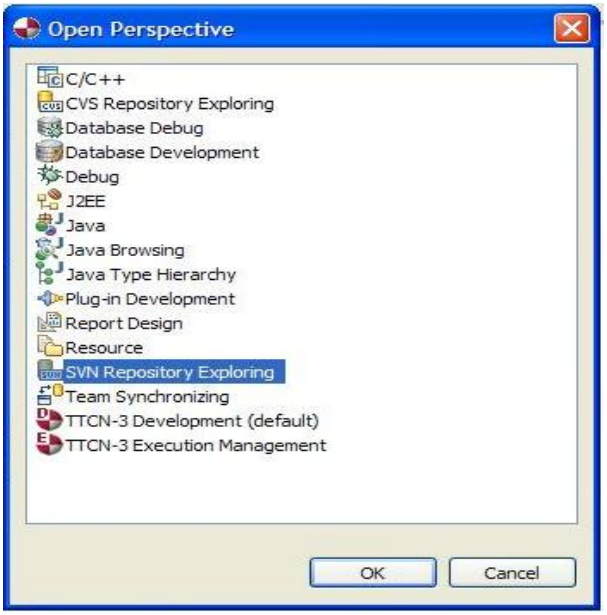
7.3 Efectivizar los cambios

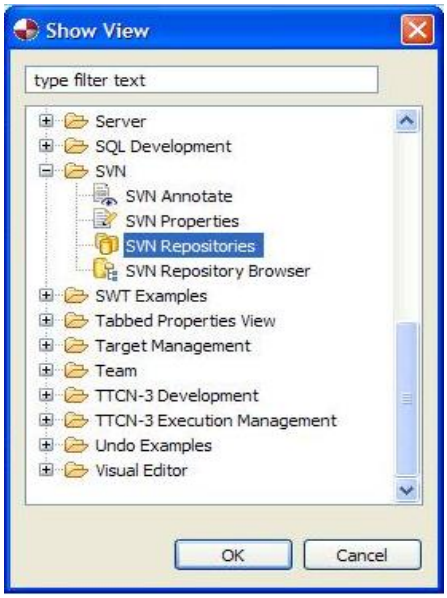
Finalizada la descarga de los archivos de actualización, aparecerá la opción para reiniciar el TTworkbench, se debe aceptar para que los cambios tengan efecto.

7.4 Configuración y utilización del Cliente SVN

Luego de reiniciado el TTworkbench se esta en condiciones de utilizar el Cliente.

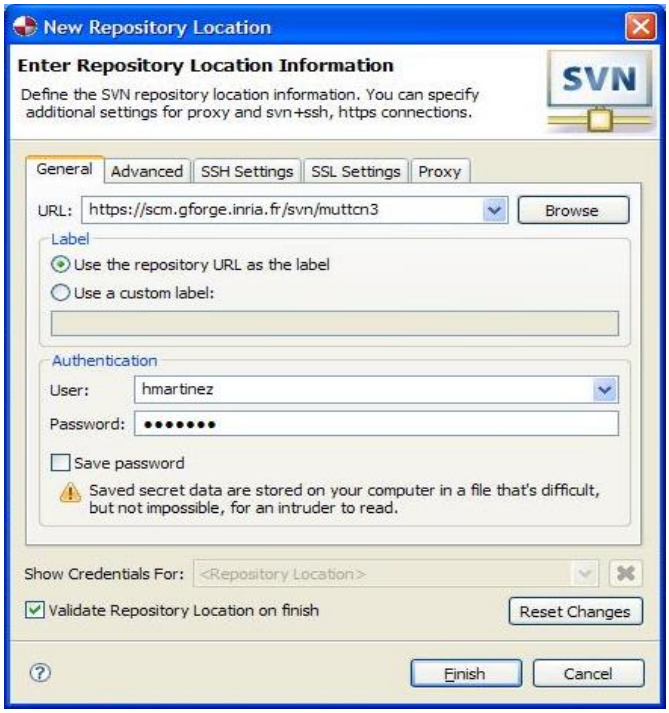
Se puede encontrar la perspectiva y las vistas en los correspondientes diálogos, que puede ser activados dentro del Menú en: *Window->Open Perspective->Other…* y *Window->Show View-> Other…* respectivamente.





Luego de abrir la vista SVN Repositories, como se mostró el figura anterior, con el botón derecho del mouse, seleccionar la opción *New->Repository Location...*

Se deberá ingresar la URL del correspondiente repositorio, el usuario asignado y la clave del mismo.



Para confirmar la sincronización hacer clic en el botón *Finish*. Después de seguir estas instrucciones se esta en condiciones de explorar el repositorio en la vista, como se muestra a continuación.

8 Referencias

1. Testing Technologies IST GmbH – TTworbench First Step User's Guide
   http://www.testingtech.com/download/users_guides/TTworkbench_1stSteps.pdf
2. Macrovision – FLEXNET LICENSING END USER GUIDE Versión 11.4
   http://www.minitab.com/uploadedFiles/Shared_Resources/Documents/License_Management/flexnet_licensing_end_user_guide.pdf
3. Polarion Community for Subversion – Subversive
   http://www.polarion.org/index.php?page=overview&project=subversive

A. Apéndice

B. Apéndice

Documentación para la instalación del Conector JNI sobre el T3DevKit.

En este apéndice se explica como instalar los módulos que integran el conector JNI. Es necesario contar con la version 0.9.1 del T3DevKit para aplicarle el parche que permite reflejar los cambios necesarios para ser enlazado con el conector y luego, configurar el adaptador Java dentro del TTwb, pudiéndose ejecutar el Test Case `DNSTester.ttcn3` incluido como ejemplo.

1 Ajustes del Sistema Operativo

Todos los pasos de instalación y configuración de este instructivo fueron llevados a cabo sobre una arquitectura de 32 bits con sistema operativo openSUSE 11.1. Para esta plataforma de referencia se creo el usuario `t3dkj` con clave `fing123` y la clave para el usuario `root` es `fing321`.

1.1 Paquetes mínimos instalados

Se debe contar con las siguientes herramientas instaladas antes de comenzar a instalar y configurar los módulos a integrar:

- `Automake` (versión 1.10.1).
- `Boost Development` (versión 1.36.0).
- `Flex` (versión 2.5.35) y `Bison` (versión 2.3).
- `gcc` y `gcc-c++` (versión 4.3).
- `Open JDK 1.6.0` (versión 1.4_b14).
- `Ant` (versión 1.7.0, recomendado).

Todas pueden ser obtenidas a través del gestor de paquetes del sistema operativo, en caso de la plataforma de referencia se utiliza `YaST`. Los números de versión son de referencia y principalmente, estas herramientas son requeridas por el T3DevKit.

1.2 Configuración de variables de entorno de Java

Se debe verificar que la variable de entorno `JAVA_HOME` este cargada con la ruta al directorio de instalación de su JDK (y no el JRE).

También es posible compilar el T3DevKit pasándole el parámetro `CCPFLAGS`, lo que genera posteriormente otros problemas. Si bien la invocación:

```
./configure CPPFLAGS="-I/usr/lib/jvm/java/include/jni.h
-I/usr/lib/jvm/java/include/linux"
```

permite la compilación, es desaconsejada.

2 Instalación del TTwb

Debe llevarse a cabo de acuerdo a las instrucciones brindadas en el Apéndice A. .

3 Instalación del T3DevKit

A continuación se brindan las instrucciones a seguir:

1) Descargar la version 0.9.1 del T3DevKit desde la página incluida en la referencia [9], el nombre del archivo comprimido es: `t3devkit-0.9.1.tar.gz`. A modo de ejemplo, se podrían ejecutar las siguientes ordenes desde línea de comandos:

```
t3dkj@ces-test:~> mkdir tmp-inst
t3dkj@ces-test:~> cd tmp-inst/
t3dkj@ces-test:~/tmp-inst> wget
http://gforge.inria.fr/frs/download.php/2473
```

2) Descomprimir el archivo en cualquier directorio, ejecutando el comando:

```
t3dkj@ces-test:~/tmp-inst> tar -zxvf t3devkit-0.9.1.tar.gz
```

3) Copiar los archivos (de la misma fuente dónde obtuvo este documento) `t3dkj-patch` y `patchgen.sh` al directorio donde se descomprimió la versión de T3DevKit.
4) Ejecutar el script `patchgen.sh`, pasándole como argumento el nombre del directorio donde se descomprimió el T3DevKit (`t3devkit-0.9.1` en este caso), ejecutando la siguiente orden:

```
t3dkj@ces-test:~/tmp-inst> ./patchgen.sh ./t3devkit-0.9.1
```

5) Ingresar al directorio donde se descomprimió el T3DevKit y ejecutar el script `autogen.sh`:

```
t3dkj@ces-test:~/tmp-inst> cd t3devkit-0.9.1/
t3dkj@ces-test:~/tmp-inst/t3devkit-0.9.1> ./autogen.sh
+ rm -f 'autom4te.cache/*'
+ aclocal
+ autoheader
+ automake --foreign --add-missing
+ autoconf
+ set +x
Now type "./configure [options]" and "make" to compile
T3DevKit.
```

6) Ejecutar el script `configure`:

```
t3dkj@ces-test:~/tmp-inst/t3devkit-0.9.1> ./configure
```

Se debe verificar que la compilación se realizará con la inclusión del TTwb, si así fuera aparecerán los siguientes mensajes:

```
checking whether to compile for Danet TTCN-3 Toolbox... no
checking whether to compile for Telelogic Tester... no
checking whether to compile for Testingtech TTCN-3... yes,
located in /home/t3dkj/tmp-inst/t3devkit-0.9.1/ttwb
```

en caso de que, la compilación no se realizara para el TTwb por defecto, se puede indicar el directorio de instalación de la herramienta con el switch:

```
--with-ttwb[=DIR]
```

También se debe verificar que se encuentran las bibliotecas de JNI, si así fuera aparecerán los siguientes mensajes:

```
checking for /usr/lib/jvm/java-1.6.0-openjdk-
1.6.0/include/jni.h... yes
```

```
checking for /usr/lib/jvm/java-1.6.0-openjdk-
1.6.0/include/linux/jni_md.h... yes
```

7) Compilar el T3DevKit ejecutando el comando `make`.
8) Instalar el T3DevKit, ejecutando el comando `make install` con permisos de administrador:

```
t3dkj@ces-test:~/tmp-inst/t3devkit-0.9.1> sudo make install
```

9) Ingresar al directorio `examples/DNSTester-ttwb`.
10) Compilar el ejemplo ejecutando el comando `make`.
11) Instalar la biblioteca generada para el ejemplo ejecutando el comando `make install` con permisos de administrador:

```
t3dkj@ces-test:~/tmp-inst/t3devkit-0.9.1/examples/DNSTester-
ttwb> sudo make install
```

12) Verificar la instalación de las bibliotecas viendo el listado de archivos en el directorio `/usr/local/lib`. Deben aparecer los siguientes archivos: `libt3cg-ttwb-DNSTester.a`, `libt3dev-ttwb.a` y `libt3dev-ttwb-dbg.a`.

4 Instalación del conector JNI

Todos los directorios incluidos en el archivo `t3conectorJNI-<ver>.tar.gz` pueden ser importados como proyectos dentro de TTwb. Para compilar los módulos C en el TTwb es necesario tener instalada el plugin de Eclipse CDT, aunque estos cuentan con archivos `makefile` propios que pueden ser ejecutados desde línea de comandos, contando solamente con el compilador `gcc`.

A continuación brindan las instrucciones a seguir:

1) Descomprimir el archivo `t3conectorJNI-<ver>.tar.gz` en cualquier directorio, ejecutando el comando:

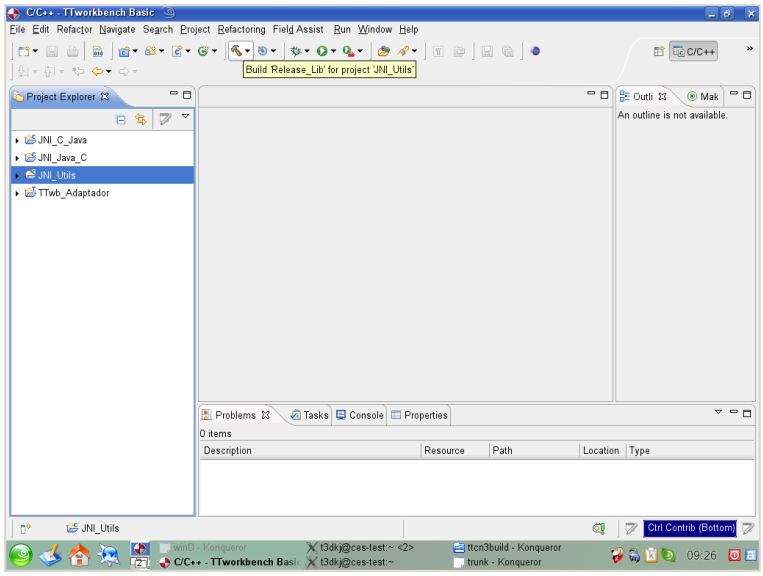
```
tar -zxvf t3conectorJNI-<ver>.tar.gz
```

En caso de que se quiera realizar la compilación de los módulos C dentro del TTwb, se deben importar los directorios `JNI_Utils`, `JNI_C_Java` y `JNI_Java_C`:

```
File → Import → General → ExistingProjects into workspace
```

Seleccionar el directorio dónde se descompactó el conector y seleccionar los 3 proyectos anteriores. Eclipse debe reconocerlos como proyectos C.

2) Compilar el módulo `JNI_Utils` ingresando al directorio `JNI_Utils/makefiles` y ejecutar el comando `make install`. Si se está dentro del TTwb, es necesario moverse a la perspectiva C/C++, seleccionar el proyecto `JNI_Utils` y presionar el botón `Build`.



3) Compilar el módulo `JNI_C_Java` ingresando al directorio `JNI_C_Java/makefiles` y ejecutar el comando make install. Si se está dentro del TTwb, es necesario moverse a la perspectiva C/C++, seleccionar el proyecto `JNI_C_Java` y presionar el botón `Build` (al igual que en el paso 2).

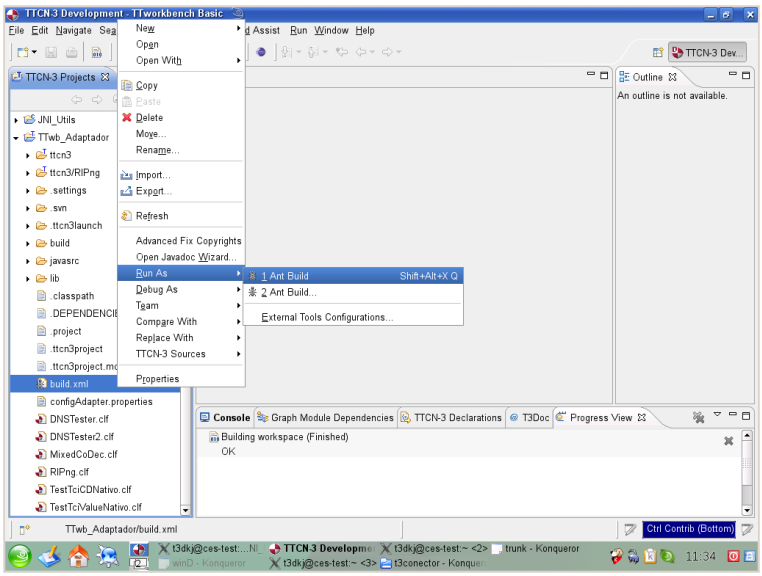
4) Compilar el módulo `JNI_Java_C` ingresando al directorio `JNI_Java_C/makefiles/nativoCG` y ejecutar el comando `make install`. Si se está dentro del TTwb, es necesario moverse a la perspectiva C/C++, seleccionar el proyecto `JNI_Java_C` y presionar el botón `Build` (al igual que en el paso 2, con la opción `Release_TTwbNativo`).

5) Desde el TTwb importar el directorio TTwb_Adaptador como proyecto:

```
File → Import → General → ExistingProjects into workspace
```

Seleccionar el directorio dónde se descompactó el conector y seleccionar el proyecto. Eclipse debe reconocerlos como proyecto TTCN-3/Java.

6) Generar el archivo `TTwb_Adaptador_TA.jar` ejecutando con `ant` el `build.xml` incluido en la raíz del proyecto (también es posible ejecutarlo desde línea de comandos, siempre y cuando esté instalada la herramienta `ant`).

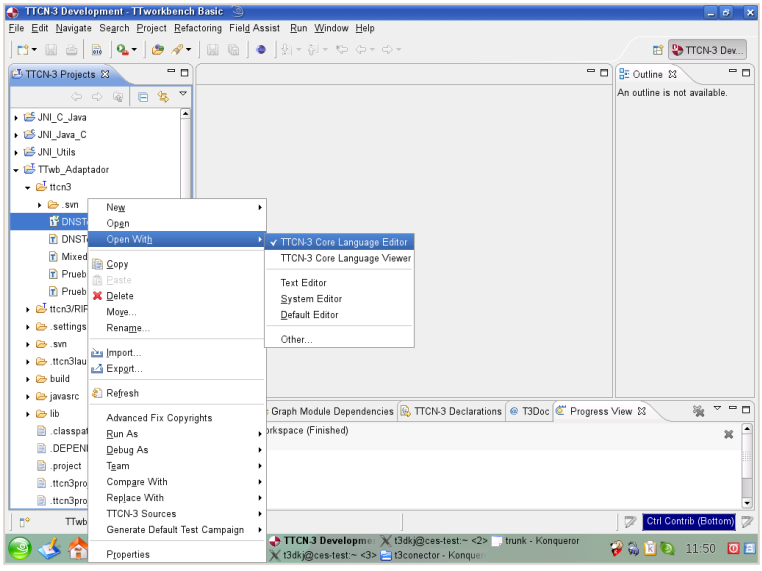


Se debe verificar que el archivo `TTwb_Adaptador_TA.jar` aparezca en el directorio `lib` del proyecto `TTwb_Adaptador`.

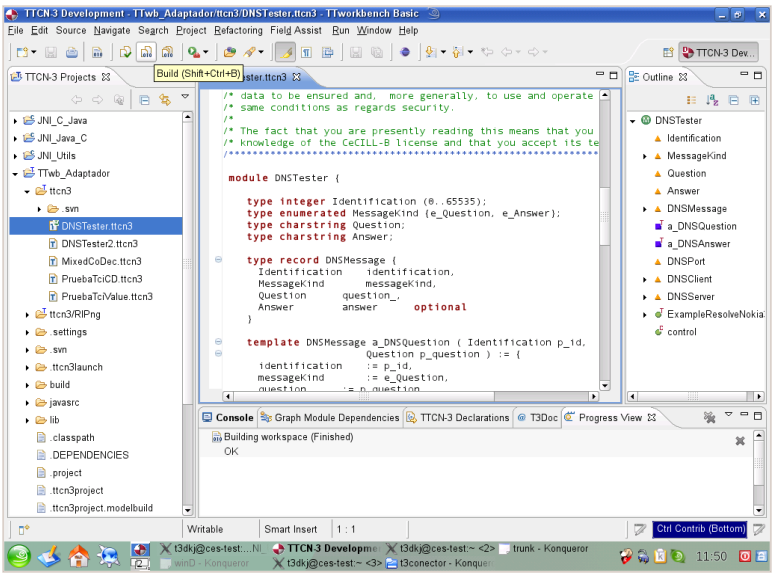
5 Ejecución del DNSTester.ttcn3

A continuación se brindan las instrucciones:

1) En la perspectiva TTCN-3 Development del TTwb, se debe buscar el fuente `DNSTester.ttcn3` dentro del proyecto `TTwb_Adaptador`; se encuentra en la carpeta `ttcn3` y debe ser abierto con el TTCN-3 Core Language Editor.

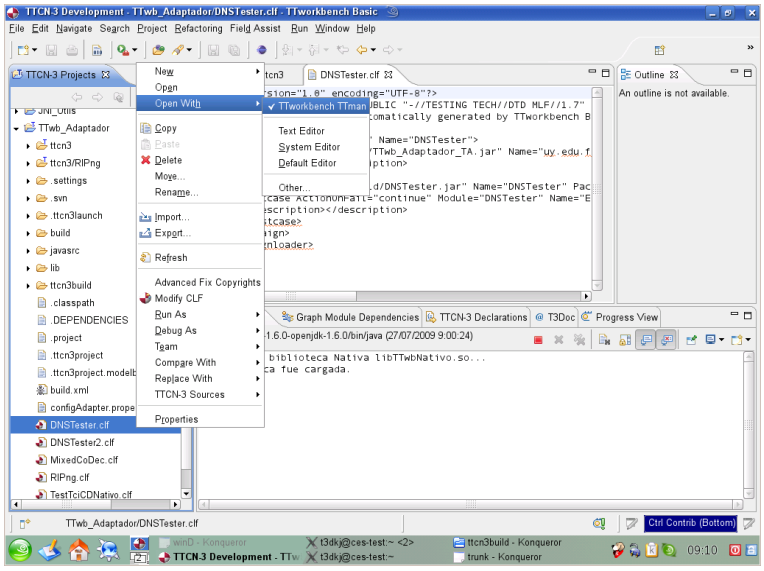


2) Compilar el fuente presionando el botón `Build`.

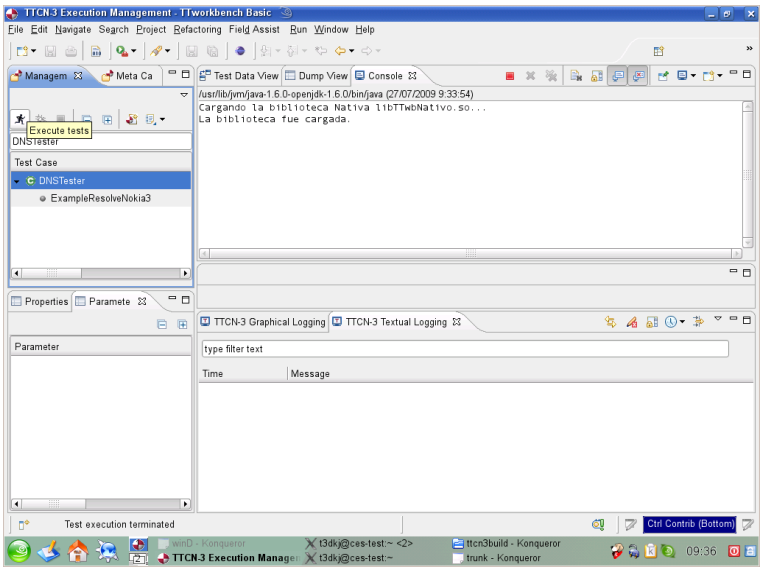


3) Se debe modificar el archivo de configuración `configAdapter.properties`, indicando el camino al archivo log y donde se encuentra la biblioteca `libTTwbNativo` generada dentro del módulo `JNI_Java_C` (por ejemplo: la ruta absoluta al directorio `JNI_Java_C/lib`).

4) Abrir el archivo `DNSTester.clf` ubicado en la raíz del proyecto `TTwb_Adaptador` con el TTworbench TTman. Esta acción realiza el cambio a la perspectiva TTCN-3 Execution Management.



5) Luego de cargado el test, oprimir el botón `Execute tests`. Esta acción comienza la ejecución y permite realizar el seguimiento de la prueba.



6 Ejecución de otros Tests Cases suministrados

Se mostrara aquí como realizar las ejecuciones para la pruebas diseñadas para en intercambio de TCI Values y la codificación/de-codificación de los mismos. Además se presentan los pasos para ejecutar los restantes Test Cases incluidos por el T3DevKit.

6.1 Test de TCI Values e integración con CDGen

A continuación se brindan las instrucciones:

1) En caso de que se quiera realizar la compilación del módulo C dentro del TTwb, se deben importar el directorios `JNI_Pruebas`

   `File → Import → General → ExistingProjects into workspace`

   Seleccionar el directorio dónde se descompactó el conector y seleccionar el proyecto. Eclipse debe reconocerlo como proyecto C.

   Para compilar el módulo `JNI_Pruebas` desde línea de comandos se debe ingresar al directorio `JNI_Pruebas/makefiles` y ejecutar el comando `make install`. Si se está dentro del TTwb, es necesario moverse a la perspectiva C/C++, seleccionar el proyecto `JNI_Pruebas` y presionar el botón `Build`.

2) Compilar el módulo `JNI_Java_C` ingresando al directorio `JNI_Java_C/makefiles/nativoTest` y ejecutar el comando `make install`. Si se esta dentro del TTwb, se necesario moverse a la perspectiva C/C++, seleccionar el proyecto `JNI_Java_C` y presionar el botón `Build` (con la opción `Release_TestNativo`).

3) Buscar y compilar los fuentes PruebaTciValues.ttcn3 y PruebaTciCD.ttcn3 análogamente a los pasos 1 y 2 descritos en el apartado 5 de este apéndice.
4) Se debe modificar el archivo de configuración `configAdapter.properties`, indicando el camino al archivo log y donde se encuentra la biblioteca `libTestNativo` generada dentro del módulo `JNI_Java_C` (por ejemplo: la ruta absoluta al directorio `JNI_Java_C/lib`).
5) Ejecutar cada uno de los tests, realizando las acciones análogas a los pasos 4 y 5 descritos en el apartado 5 de este apéndice.

6.2 Otros ejemplos incluidos en el T3DevKit

A continuación se brindan las instrucciones:

1) Compilar los ejemplos ubicados en los directorios `DNSTester2-ttwb`, `MixedCoDec-ttwb`, y `RIPng-ttwb`, siguiendo análogamente los pasos 10 al 12 descritos en el apartado 3 de este apéndice.
2) Verificar la instalación de las bibliotecas viendo el listado de archivos en el directorio `/usr/local/lib`. Deben aparecer los siguientes archivos: `libt3cg-ttwb-DNSTester2.a`, `libt3cg-ttwb-MixedCoDec.a` y `libt3cg-ttwb-RIPng.a`.
3) Se debe re-compilar el módulo JNI_Java_C como se describió en el paso 4 del apartado 4 de este apéndice, después de cambiar el makefile ubicado en el directorio `makefiles/nativoCG` para realizar el enlace con la biblioteca correspondiente al test que se quiera ejecutar.
4) Ajustar la configuración como se mostró en el paso 3 del apartado 5 de este apéndice.
5) Ejecutar cada uno de los tests, realizando las acciones análogas a los pasos 4 y 5 descritos en el apartado 5 de este apéndice.

B. Apéndice

C. Apéndice

Seguimiento del test para TCI Values

Luego de ejecutado el módulo TTCN-3 específicamente diseñado para el envío de valores TCI al conector, se obtuvo la traza descrita en este apéndice, mostrando la salida en al consola Java y lo escrito en el archivo de logging para el wrapper.

Salida por consola de la ejecución del test de TCI Values:

```
Cargando la biblioteca Nativa libTestNativo.so...
La biblioteca fue cargada.
***** Java TciCDRequired *****
Resultado para Tipos: true

***** Java TciValue *****
Resultado para Value: true
Address: NO se conoce la clase que lo representa
Resultado para CharString: true
Resultado para BitString: true
Resultado para Boolean: true
Resultado para Enumerated: true
Resultado para Float: true
Resultado para HexString: true
Resultado para Integer: true
Resultado para ObjId: true
Resultado para OctectString: true
Resultado para Record: true
Resultado para Set: true
Resultado para RecordOf: true
```

```
Resultado para SetOf: true
Resultado para UniversalCharString: NO esta soportado en la implementacion nativa
Resultado para Union: true
Resultado para Verdict: true
```

Archivo de Logging para la ejecución del test de TCI Values:

```
[2009-06-30 02:59:06] - Inicio de la ejecucion...
[2009-06-30 02:59:17] - ************** TCIREQUIRED - TYPES **************
[2009-06-30 02:59:17] -    Tipo por Nombre: [integer]
[2009-06-30 02:59:17] -    Tipo Integer: [integer]
[2009-06-30 02:59:17] -    Tipo Float: [float]
[2009-06-30 02:59:17] -    Tipo Boolean: [boolean]
[2009-06-30 02:59:17] -    Tipo Char: [char]
[2009-06-30 02:59:17] -    Tipo UChar: [NO implementado]
[2009-06-30 02:59:17] -    Tipo Objid: [objid]
[2009-06-30 02:59:17] -    Tipo Charstring: [charstring]
[2009-06-30 02:59:17] -    Tipo UCharstring: [universal charstring]
[2009-06-30 02:59:17] -    Tipo Hexstring: [hexstring]
[2009-06-30 02:59:17] -    Tipo Bitstring: [bitstring]
[2009-06-30 02:59:17] -    Tipo Octetstring: [octetstring]
[2009-06-30 02:59:17] -    Tipo Verdict: [verdicttype]
[2009-06-30 02:59:17] - ************** VALUE - TYPE **************
[2009-06-30 02:59:17] -    No presente: 0
[2009-06-30 02:59:17] -    Value Codificacion: [(null)]
[2009-06-30 02:59:17] -    Value Codificacion Variante: [(null)]
[2009-06-30 02:59:17] -    Tipo: 10
[2009-06-30 02:59:17] -    Nombre: [AddressValue]
[2009-06-30 02:59:17] -    Modulo:
[2009-06-30 02:59:17] -           Nombre: [PruebaTciValue]
[2009-06-30 02:59:17] -           Objeto: [PruebaTciValue]
[2009-06-30 02:59:17] -           Auxiliar: 0
[2009-06-30 02:59:17] -    Type Codificacion: [(null)]
[2009-06-30 02:59:17] -    Type Codificacion Variante: [(null)]
[2009-06-30 02:59:17] -    Nueva instancia de: AddressValue
[2009-06-30 02:59:17] -    Type Extension: []
[2009-06-30 02:59:17] - ************** CHARSTRINGVALUE **************
[2009-06-30 02:59:17] -    Numero de Caracteres: 6
[2009-06-30 02:59:17] -    Contenido: [cad3na]
[2009-06-30 02:59:17] -    Caracter: 3
[2009-06-30 02:59:17] - ************** BITSTRINGVALUE **************
[2009-06-30 02:59:17] -    Numero de Bits: 5
[2009-06-30 02:59:17] -    Contenido: ['10011'B]
[2009-06-30 02:59:17] -    Bit: 1
```

```
[2009-06-30 02:59:17] - ************** BOOLEANVALUE **************
[2009-06-30 02:59:17] -    Valor Boolean: 0
[2009-06-30 02:59:17] - ************** ENUMERATEDVALUE **************
[2009-06-30 02:59:17] -    Enumerado: clase2
[2009-06-30 02:59:17] - ************** FLOATVALUE **************
[2009-06-30 02:59:17] -    Valor Float: 54.750000
[2009-06-30 02:59:17] - ************** HEXSTRINGVALUE **************
[2009-06-30 02:59:17] -    Numero de Hex: 5
[2009-06-30 02:59:17] -    Contenido: ['AE0C8'H]
[2009-06-30 02:59:17] -    Hex: 12
[2009-06-30 02:59:17] - ************** INTEGERVALUE **************
[2009-06-30 02:59:17] -    Valor Absoluto: [19345]
[2009-06-30 02:59:17] -    Numero de Digitos: 5
[2009-06-30 02:59:17] -    Positivo: 0
[2009-06-30 02:59:17] -    Digito: 9
[2009-06-30 02:59:17] - ************** OBJIDVALUE **************
[2009-06-30 02:59:17] -    Numero de Elementos: 2
[2009-06-30 02:59:17] -    Elementos:
[2009-06-30 02:59:17] -           ASCII: [12345]
[2009-06-30 02:59:17] -           Numero: 12345
[2009-06-30 02:59:17] -           Auxiliar: 0
[2009-06-30 02:59:17] -
[2009-06-30 02:59:17] -           ASCII: [67890]
[2009-06-30 02:59:17] -           Numero: 67890
[2009-06-30 02:59:17] -           Auxiliar: 0
[2009-06-30 02:59:17] -
[2009-06-30 02:59:17] - ************** OCTETSTRINGVALUE **************
[2009-06-30 02:59:17] -    Numero de Octet: 2
[2009-06-30 02:59:17] -    Contenido: ['0426'O]
[2009-06-30 02:59:17] -    Octet: 4
[2009-06-30 02:59:17] - ************** RECORDVALUE **************
[2009-06-30 02:59:17] -    Campo 'numero': 1234
[2009-06-30 02:59:17] -    Nombres de Campos:
[2009-06-30 02:59:17] -           [nombre]
[2009-06-30 02:59:17] -           [numero]
[2009-06-30 02:59:17] -           [clase]
[2009-06-30 02:59:17] - ************** SETVALUE **************
[2009-06-30 02:59:17] -    Campo 'numero': 1234
[2009-06-30 02:59:17] -    Nombres de Campos:
[2009-06-30 02:59:17] -           [nombre]
[2009-06-30 02:59:17] -           [numero]
[2009-06-30 02:59:17] -           [clase]
[2009-06-30 02:59:17] - ************** RECORDOFVALUE **************
```

```
[2009-06-30 02:59:17] -    Tipo de elementos: integer
[2009-06-30 02:59:17] -    Largo actual: 4
[2009-06-30 02:59:17] -    Elementos nuevo listado:
[2009-06-30 02:59:17] -            [-1]
[2009-06-30 02:59:17] -            [-1]
[2009-06-30 02:59:17] - ************** SETOFVALUE **************
[2009-06-30 02:59:17] -    Tipo de elementos: integer
[2009-06-30 02:59:17] -    Largo actual: 4
[2009-06-30 02:59:17] -    Elementos nuevo listado:
[2009-06-30 02:59:17] -            [+4]
[2009-06-30 02:59:17] -            [+4]
[2009-06-30 02:59:17] - ************** UNIVERSALCHARSTRINGVALUE **************
[2009-06-30 02:59:17] - ************** UNIONVALUE **************
[2009-06-30 02:59:17] -    Nombre de Variante Presente: numero
[2009-06-30 02:59:17] -    Variante 'numero': 5678
[2009-06-30 02:59:17] -    Nombres de Variantes:
[2009-06-30 02:59:17] -            [numero]
[2009-06-30 02:59:17] -            [cadena]
[2009-06-30 02:59:17] - ************** VERDICTVALUE **************
[2009-06-30 02:59:17] -    Veredicto: 1
```

D. Apéndice

Seguimiento del test para el Codec

Luego de ejecutado el módulo TTCN-3 diseñado para el envío de valores TCI a ser codificado y de-codificados utilizando el conector, se obtuvo la traza descrita en este apéndice, mostrando la salida en al consola Java y lo escrito en el archivo de logging para el wrapper.

Salida por consola de la ejecución del test de Codec:

```
Cargando la biblioteca Nativa libTTwbNativo.so...
La biblioteca fue cargada.
```

Archivo de Logging para la ejecución del test de Codec:

```
[2009-06-30 19:20:57] - Inicio de la ejecucion...
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 19:21:00] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
```

D. Apéndice

E. Apéndice

Seguimiento del test DNSTester

Luego de ejecutado el módulo TTCN-3 DNSTester incluido en los ejemplos del T3DevKit, el cual necesita de la interacción entre las plataformas Java y C/C++ mediante el uso del conector, se obtuvo la traza descrita en este apéndice, mostrando la salida en al consola Java y lo escrito en el archivo de logging para el wrapper.

Salida por consola de la ejecución del test de Codec:

```
Cargando la biblioteca Nativa libTTwbNativo.so...
La biblioteca fue cargada.
DNSPort: using 200.58.141.65:53 as remote DNS server
```

Archivo de Logging para la ejecución del test de Codec:

```
[2009-06-30 20:39:36] - Inicio de la ejecucion...
[2009-06-30 20:39:38] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triExecuteTestcase
*****
[2009-06-30 20:39:38] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triMap *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tci_cd_Codec_encode *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triSend *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStartTimer
*****
[2009-06-30 20:39:39] - ########## triEnqueueMsg ##########


[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tci_cd_Codec_decode *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triTimerRunning
*****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triReadTimer *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triUnmap *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triEndTestCase
*****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triSAReset *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triPAReset *****
```

```
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-06-30 20:39:39] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
```

F. Apéndice

Seguimiento del test RIPng

Luego de ejecutado el módulo TTCN-3 RIPng incluido en los ejemplos del T3DevKit, el cual necesita de la interacción entre las plataformas Java y C/C++ mediante el uso del conector, se obtuvo la traza descrita en este apéndice, mostrando la salida en al consola Java y lo escrito en el archivo de logging para el wrapper.

Salida por consola de la ejecución del test de Codec:

```
Cargando la biblioteca Nativa libTTwbNativo.so...
La biblioteca fue cargada.
```

Archivo de Logging para la ejecución del test de Codec:

```
[2009-07-06 19:13:25] - Inicio de la ejecucion...
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triExecuteTestcase
*****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triMap *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triMap *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triMap *****
[2009-07-06 19:14:25] - *****
Java_uy_edu_fing_tri_adapter_Adapter_triExternalFunction *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triEndTestCase
*****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triSAReset *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triPAReset *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
```

```
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
[2009-07-06 19:14:25] - ***** Java_uy_edu_fing_tri_adapter_Adapter_triStopTimer *****
```

G. Apéndice

En este apartado anexo, se presenta el contenido de la carta presentada a los evaluadores y organizadores de la T3UC 2008, con el fin de que ser tenidos en cuenta para la obtención de una beca. La misma fue otorgada e incluyó la participación en los tutoriales, la conferencia y del apoyo económico para poder asistir.

TTCN-3 User Conference 2008 – Write-up

I am a student of Grade, currently performing my thesis final in the "Facultad de Ingeniería - Universidad de la República – Uruguay", supervised by Ariel Sabiguero Pedeciba/UDELAR.

The motivation of our project was to generate a piece of software that allows the reuse of existing components, promoting interoperability between platforms Java and C++ and also achieve a high level maintainability of the products.

Given that, there is the TestingTech Java compiler and tool TTCN-3 Codec Generator C/C++ of IRISA, the solution designed consists of a JNI connector, by implementing native interfaces defined by Go4IT based standards ETSI, plus the use of the features included in T3DevKit provides generic components Adapter and Codec required; those that allow the execution of test cases written in language TTCN-3.

It is worth mentioning, that the T3DevKit is not a tool independent of the implementation of interfaces to provide for compiler TTCN-3 being used, which include as an objective analysis of the impacts generated by the use of Java compiler mentioned, and this led to various adaptations to be made and additions to the tool to achieve the appropriate integration without affecting the functionality already available.

Another interesting point to comply with the construction of the connector, is the review of implementation of TTCN-3 types, this applies to objects used by the Java compiler, as well as the definitions of types C/C++ as defined in the interfaces.

The responsibility of the JNI connector is to deploy them natively interfaces Java Provided (for TCI/CD, TRI/SA TRI/PA) reusing functionality offered by the T3DevKit and then allow the invocation from C/C++ interfaces Required (for TCI/CD, TRI/SA TRI/PA) implemented in Java, they are used to generate the transfer of objects from the T3DevKit.

G. Apéndice

It is of my interest to know more tools existing TTCN-3, the types of solutions in which it applies, as well as the way in which the various adaptations are resolved, particularly the variety of libraries in the different platforms. They are also of interest to me, future projections on those tools and technological trends of the solutions, as well as trends in the language itself. Thus, it would be very useful for me to participate in the TTCN-3 User conference 2008, knowing and being in contact with the community of professionals and researchers in TTCN-3. Searching gain knowledge that can help me complete my academic work, as well as, hopefully input thinking about my participation in the INRIA International Internship - “T3DevKit and mu-TTCN-3 unified parsing”.