PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Maestría
en Informática

Metodologías y herramientas para testing

Construcción de un compilador TTCN-3 con

soporte multiplataforma

Ricardo Rezzano Rumi

2015

Ricardo Rezzano Rumi
Metodologías y herramientas para testing
Construcción de un compilador TTCN-3
con soporte multiplataforma
ISSN 0797-6410
Tesis de Maestría en Informática
Reporte Técnico RT 15-05
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, 2015

PEDECIBA Informática

Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Metodologías y Herramientas para Testing Construcción de un compilador TTCN-3 con soporte multiplataforma

Montevideo, 13 de Diciembre de 2011

Tesis de Maestría Presentada por: Ricardo Rezzano Rumi
rrezzano@fing.edu.uy

Directores Académicos:

PEDECIBA-InCo Héctor Cancela
cancela@fing.edu.uy

PEDECIBA-InCo Ariel Sabiguero
asabigue@fing.edu.uy

Directores de Tesis: Juan J. Cabezas
Ariel Sabiguero

Tribunal: Javier Baliosian
Ana Cavalli
Nora Szasz

Metodologías y Herramientas para Testing
Construcción de un compilador TTCN-3 con soporte multiplataforma

Resumen:

TTCN-3 [1], el lenguaje de testing estandarizado por ETSI, ofrece una alta capacidad de abstracción para la especificación de Casos de Prueba (Test Cases) y es ampliamente usado en las áreas de pruebas de conformidad, especialmente aplicado a protocolos de red.

Este trabajo se propone abordar algunas necesidades del lenguaje TTCN-3, detectadas en el área de testing de conformidad para protocolos de red, cuya solución entendemos promoverá su uso y el de las plataformas de prueba de la industria construidas en torno al mismo.

Para dar soporte a este proceso, en el marco de un proyecto con universidades de varios países, construimos una plataforma abierta y libre para la experimentación en TTCN-3, la cual se compone de un compilador TTCN-3, su sistema de tiempo de ejecución y la aplicación de gestión de los casos de pruebas. Si bien la plataforma desarrollada sirvió inicialmente para realizar los experimentos necesarios a nuestra investigación, es decir validar experimentalmente las propuestas para la solución de alguno de los problemas planteados, la misma servirá en el futuro para realizar otros experimentos, entrenamiento y estudio sobre el lenguaje.

El tema concreto que nos propusimos solucionar fue la interoperabilidad y portabilidad de las herramientas, bibliotecas e implementaciones en TTCN-3 entre las diferentes plataformas de base, que este usa para interactuar con los sistemas a probar a través de las interfaces definidas por el estándar a estos efectos.

El estándar, que define el lenguaje TTCN-3, incluye mapeos a dos plataformas de base para la implementación de sus interfaces, Java y C/C++. Cuando comenzamos la investigación y hasta el momento de publicar nuestros resultados, las herramientas y compiladores de TTCN-3, en general, se especializaban en una de las plataformas, pero dejaban de lado la otra, afectando negativamente las capacidades de reuso entre ellas. En la práctica solo algunos casos de prueba, que abarcan áreas específicas de aplicación, pueden ser implementados eficientemente usando un único lenguaje de plataforma, mientras otras áreas son abordadas mas eficientemente desde el otro. Si bien esta situación cumple con la especificación del lenguaje no es la única interpretación posible de los estándares que lo definen. La alternativa que planteamos es disponer de un compilador que implemente el mapeo de las interfaces para ambas plataformas al mismo tiempo, soportando de forma directa la integración con estas.

Esta tesis describe el proceso de modelado y construcción del compilador TTCN-3 y del proceso de investigación y definición de la solución encontrada para el tema de la portabilidad entre las plataformas de base, también presenta la prueba de conceptos realizada para validar la misma.

Palabras claves: Testing, Prueba, TTCN-3, Compilador, Interoperabilidad, Portabilidad, API-Dual

[1] **Test and Testing Control Notation version 3**, en su tercera versión el lenguaje definido por el estándar de ETSI **873-1 Part 1** es un lenguaje de testing que se aplica tanto a pruebas de conformidad de protocolos como a todo tipo de sistemas reactivos, mediante la definición de diferentes tipos de puertos [14]

Índice general

Índice de figuras

Índice de tablas

Introducción

Uno de los desafíos permanentes de la Industria del Software es asegurar los niveles requeridos en la calidad de sus productos, de forma de grantizar que las funciones realizadas por éstos se ejecuten de acuerdo a las especificaciones correspondientes y por otro lado que estas correspondan al comportamiento esperado del sistema. Existen un conjunto de actividades tendientes a asegurar estos niveles de calidad, usualmente denominadas verificación y validación (V&V), que se aplican durante todo el proceso de desarrollo del software. El Testing continua siendo una de las actividades más importantes de este grupo, es decir la realización de diferentes tipos de pruebas del software para verificar el comportamiento del mismo en unas condiciones determinadas. Las posibles pruebas a realizar pueden clasificarse de varias formas, entre ellas por el tipo de características del software que se prueba, así tenemos pruebas funcionales, de conformidad, desempeño, robustez, etc; otra clasificación puede ser respecto al alcance de la prueba, acá tenemos pruebas unitarias o de módulos, de integración, de sistemas, etc. [30].

Existen diferentes metodologías a aplicar para la realización de las pruebas, una importante división en este sentido se da entre la realización de pruebas en forma manual o la automatización de las mismas mediante el uso de un software. Mientras las pruebas manuales son claramente menos costosas de implementar en primera instancia, estas tienden a mantener un importante costo de realización a lo largo del tiempo. En cambio las pruebas automáticas generalmente tienen un costo inicial mayor, relacionado a la implmentación del sistema de prueba, pero su repetición basada en la utilización de dicho sistema es menos costosa, por lo cual su costo total disminuye con la cantidad de repeticiones o el tiempo durante el cual las mismas son realizadas. Depende entonces, en gran medida, de los sistemas a probar que justifiquen o no incurrir en el costo de automatizar los procesos de prueba, su dimensión y el hecho de que se prevea la existencia de nuevas versiones del mismo, incluso la naturaleza de los mismos es decir si son soluciones a medida o productos que serán utilizados extensamente en diferentes instalaciones y contextos operativos. Todos estos factores considerados en conjunto, determinaran si se justifica invertir en la automatización de las pruebas. Aquellos donde se justifica la automatización de las pruebas, se caracterizan por tener la necesidad de repetir las mismas, de forma relativamente similar, para una cantidad grande de implementaciones o para soluciones de distintos fábricantes, aquí se encuentran por ejemplo, las pruebas de conformidad para diferentes dispositivos basados en software o las pruebas de productos en sus diferentes versiones, también conocidas como pruebas de regresión. El análisis de estos costos es una materia de por si compleja y escapa al alcance de nuestro trabajo, pero el mismo es abordado con variadas perspectivas en diferentes trabajos de la industria y la academia. [7, 29]

Dentro de este tipo de procesos de prueba se presentan dos alternativas para su implementación, una es realizar un desarrollo *ad hoc* para la automatización de las pruebas, la otra es recurrir a una plataforma o herramienta para el desarrollo y automatización de pruebas, también en este caso nos enfrentamos a una decisión basada en la relación costo-beneficio de las alternativas planteadas. En el segundo caso, que es el que nos ocupa, el costo de la realización de pruebas es aún relativamente

alto debido, entre otros factores, al estado del arte de las tecnologías y herramientas disponibles en el mercado para automátizar las pruebas y el costo asociado a los procesos posteriores de automatización de las pruebas, es decir la imlemetación del sistema de testing.

TTCN-3 es una herramienta de este último tipo, la cual es usada especialmente en pruebas de conformidad para protocolos de red, aunque no se limita a estas. Esta herramienta, su estudio, evaluación y propuestas de mejoras a la misma, serán en gran medida el principal asunto de nuestro trabajo.

El estándar ISO/IEC 9646 "Conformance testing methodology and framework" (CTMF) en su parte III [28], definió originalmente una notación para la especificación de pruebas TTCN, **Tree and Tabular Combined Notation**, inicialmente concebido para el testing de protocolos OSI ha evolucionado posteriormente hacia otros ámbitos. El mismo ha sido promovido y desarrollado por ETSI [9] desde su inicio, para la especificación e implementación de pruebas de sistemas en general, actualmente se ha convertido en el estándar de ETSI **ETSI ES 202** en su liberación 4.1.1. En su tercera versión TTCN es un lenguaje flexible y potente diseñado para crear todo tipo pruebas sobre sistemas reactivos, el cual abarca desde la especificación abstracta de los casos de prueba hasta la creación de los ejecutables para los mismos. Los cambios realizados han alcanzado a su nombre que manteniendo la sigla TTCN ha cambiado su significado a **Testing and Test Control Notation** [14, 10, 11, 12, 15, 13]. En la definición y diseño de TTCN-3 se tuvieron en cuenta elementos tecnológicos y funcionales que han evolucionado el lenguaje de forma significativa, orientándolo a la realización de pruebas en general y utilizando paradigmas recientes de los lenguajes de programación, para convertirlo en una herramienta más usable y eficiente en la creación de casos de prueba.

Uno de los atractivos de TTCN-3 es formar parte del estándar que lo define, el cual abarca tanto la definición del lenguaje en si mismo, como la metodología para la definición, implementación y realización de las pruebas, el hecho de que este sea un estándar de ETSI es también un elemento importante para la promoción y uso del lenguaje.

El estándar TTCN ya cuenta con 3 grandes versiones y varias implementaciones del lenguaje en diferentes plataformas, realizadas por diferentes fabricantes, contando la última versión con más de diez años de experiencia en la industria. Estas implementaciones están especializadas en diferentes nichos de aplicación y plataformas tecnológicas. Esto le ha brindado diversidad en las áreas de aplicación, madurez y desarrollo al lenguaje y ha generado nuevos requerimientos en cada una de sus nuevas versiones. De esta forma se constituye en una herramienta de testing con una vasta experiencia, un importante respaldo teórico y formal, así como variados trabajos de investigación que lo proyectan en nuevas áreas y aplicaciones, estos elementos lo convierten en una interesante opción al evaluar herramientas para la automatización de pruebas.

Las áreas tradicionales de aplicación de TTCN-3 han sido: prueba de protocolos, prueba de módulos, prueba de APIs y prueba de servicios; aunque las características actuales del mismo permiten realizar pruebas sobre cualquier sistema reactivo. Respecto a los tipos de pruebas, si bien TTCN-3 es ampliamente utilizado en pruebas de conformidad, este puede utilizarse en cualquier tipo de pruebas, tal como se ha presentado en las conferencias del área, incluyendo pruebas funcionales, pruebas de interoperabilidad, robustez, regresión, sistemas, integración y otras.

En nuestro caso, la principal experiencia concreta en el lenguaje del equipo de ARMOR[4]/IRISA[27], al que nos vinculamos para realizar este trabajo, está concentrada en la realización de pruebas de conformidad de dispositivos de red para la nueva versión del protocolo de Internet:IPv6.

Más allá de estos avances algunos aspectos permanecen aún pendientes de resolverse, entre estos destacamos la necesisdad de independizar completamente la defincón de los casos de prueba de las plataformas de base y el enriquecimiento de los mecanismos de comparación (matching) para los mensajes intercambiados con los sistemas bajo prueba. Otro aspecto donde el lenguaje

presenta algunas debilidades es en su penetración en el mercado más allá de sus áreas de aplicación tradicionales, esta puede explicarse en parte por los costos que involucra su uso para escalas más pequeñas.

Para profundizar los aspectos relacionados al uso y necesidades actuales de TTCN-3 tomamos como referencia los trabajos presentados en las conferencias de usuarios del lenguaje, TTCN-3 UC (TTCN-3 User Conference) de los últimos años, donde pueden encontrarse desarrollados cada uno de estos. [17, 18, 19]

Nuestro interés se enfocó en la adquicisión de conocimientos y la realización de investigación y su correspondiente experimentación en el área de testing, especialmente en los temas relacionados con el estándar para testing TTCN-3 y la promoción de su uso como plataforma para la realización de pruebas, tanto en sus áreas de aplicación tradicional, especialmente en pruebas de conformidad para protocolos de red, como en nuevas áreas. Parte importante de nuestro trabajo se vió influido por encontrarnos en el contexto de las actividades de GTTP's (Testing Protocols Team Work - Grupo de Trabajo en Testing de Protocolos), desde el InCo - Fing de la Universidad de la República (Uruguay), en colaboración con ARMOR/IRISA.

Objetivos Generales

El objetivo principal de este trabajo es aportar propuestas para solucionar algunos de los aspectos considerados como pendientes o abiertos en la interpretación del estándar y la implementación del lenguaje TTCN-3, especialmente en aspectos relacionados a las pruebas de protocolos de red, así como comprobar experimentalmente la factibilidad de dichas propuestas.

Para poder identificar estos problemas y aportar soluciones se requería adquirir un conocimiento relativamente profundo del lenguaje y recoger las conclusiones de la experiencia acumulada en la industria, en el uso de las diferentes herramientas basadas en el estándar para diferentes áreas de aplicación, aunque priorizando aquellas relacionadas con nuestra área de interés.

Por otra parte, para poder experimentar y probar las soluciones propuestas, se requería tener acceso a una implementación concreta de la plataforma TTCN-3, incluyendo la capacidad para modificarla, esto nos llevó a considerar como objetivo intermedio desarrollar una plataforma experimental del lenguaje.

Al definir estos objetivos las actividades para trabajar en torno a estos quedaron organizadas en tres etapas. En primer termino la adquisición de conocimiento del área de Testing, las tecnologías y herramientas disponibles, especialmente las vinculadas a TTCN-3 y la experiencia de prueba en protocolos para redes, que describimos en esta introducción y el **Capítulo 1, Características del lenguaje TTCN-3**. En segundo termino la construcción de una plataforma de experimentación para TTCN-3 que desarrollamos en la **Parte I, El Compilador**. Por último, una vez que contamos con la plataforma TTCN-3 procedimos a investigar, definir y experimentar en ella soluciones al tema de interoperabilidad y portabilidad con las platarformas de base, que entendímos de valor para promover el uso de TTCN-3, trabajo al que nos dedicaremos en la **Parte II, La Dualización**.

Construcción de un compilador experimental para TTCN-3

Uno de los desafíos enfrentados para lograr implementar pruebas de concepto para las ideas a proponer, fue disponer de una plataforma TTCN-3 de experimentación, es decír la posibilidad de contar con un compilador TTCN-3, no solo para su uso, sino también para poder acceder a sus fuentes y contar con el derecho a modificarlos. Obtener una plataforma de la industria en estas condicones no era algo sencillo, nuestra experiencia y también la de otros equipos de IRISA, indicaba que para disponer plataformas TTCN-3, solo a efectos de su uso, el proceso era lento y

no se podía asegurar el resultado. Como alternativa a esta situación nos propusimos construir una implementación del compilador TTCN-3 con un alcance restringido. Este proyecto fue denominado en primera instancia μTTCN-3.

El alcance de este trabajo estuvo determinado por el objetivo de lograr una versión operativa del compilador, la cual debía cubrir las áreas del lenguaje más usadas, especialmente aquellas relevantes a efectos de los experimentos planteados, pero que podía dejar fuera caractéristicas que no fuesen estríctamente necesarias en esta etapa y que requeriesen un esfuerzo importante para su desarrollo.

Este proyecto para la construcción de una implementación del compilador TTCN-3, se integró al proyecto Go4IT [22], el cual incorporó el trabajo de múltiples equipos en diferentes países responsables por diversas áreas del proceso de construcción, en ese contexto nuestra responsabilidad específica fue el diseño general del compilador, el diseño y desarrollo del Analizador Sintáctico y Léxico y la orientación en el equipo responsable de la traducción.

Como resultado de este trabajo quedó disponible a fines del anõ 2007 la primera versión del compilador abierto y libre de TTCN-3, la **Versión A0** del compilador fue renombrada en este momento como ρTTCN-3.

Como parte de los objetivos propuestos en la construcción del compilador, éste proceso nos aporto un nivel de conocimiento sobre el lenguaje que nos facilitó entender varios de los problemas experimentados por los desarrolladores y nos capacitó para proponer soluciones a algunos de estos. Además, la existencia del compilador permitió en la siguiente etapa, como nos lo habíamos planteado, experimentar la solución propuesta realizando las extensiones necesarias a ρTTCN-3

Motivación de la solución Dual

Elegimos como la principal área de interés para la realización de aportes, la portabilidad e interoperabilidad de las soluciones, bibliotecas y herramientas TTCN-3 entre las plataformas de base soportadas por el estándar, entendiendo que de esta forma promoveríamos el uso del lenguaje al facilitar el reuso de implementaciones y bibliotecas entre diferentes productos y plataformas.

En líneas generales la situación planteada es que, si bien existen muy buenos productos de Testing que implementan el estándar TTCN-3, cada uno de ellos está enfocado en diferentes nichos del mercado, con fortalezas específicas para estos pero limitándose a una única plataforma de base.

Como resultado de estas características en los productos y la metodología asociada a los mismos, se arriba a una situación *de facto* donde cada comunidad trabaja de forma relativamente aislada, desalentando el reuso de bibliotecas y herramientas entre ellas, al momento de implementar el mismo tipo de Casos de Prueba o incluso los mismos Casos de Prueba.

Esta situación tiene especial impacto en las áreas de Testing muy especializadas, que son naturalmente más fáciles de abordar desde una de las plataformas, siendo significativamente más costoso hacerlo desde la otra plataforma. Es en estos casos donde resultaría más valioso poder usar los Adaptadores de la otra plataforma de una manera sencilla y eficiente. Como ejemplos de esta situación podemos presentar los protocolos de red, que naturalmente son desarrollados en **C/C++** y los Web Services que se adaptan más fácilmente desde las plataformas **Java**.

La selección de una herramienta fuerza a los desarrolladores y por lo tanto a las empresas a pagar el sobrecosto del cruce de plataformas, cada vez que esto es necesario o a utilizar una herramienta de desarrollo que no es la más adecuada para la tarea que se debe realizar, con el consiguiente sobrecosto al momento de desarrollar el Adaptador correspondiente.

El estándar no impone que la herramienta deba estar basada en **Java** o **C** exclusivamente, por lo cual esta situación *de facto* no es requerida por el estándar, sino que está generada por razones técnicas y prácticas. Las herramientas al momento de usar las implementaciones y bibliotecas en la

plataforma de base dependen de los mecanismos nativos para el enlazado de los mismos, esto tiene como consecuencia que estas se especialicen en una de las plataformas.

Como solución a esta situación nos propusimos considerar la alternativa de que las herramientas TTCN-3 sean capaces de interoperar con bibliotecas y herramientas en ambas plataformas en forma directa y transparente para el desarrollador de TTCN-3, independizando de esta forma las plataformas de base de los Adaptadores y otras bibliotecas, usadas en las implementaciones, de la plataforma de base de la herramienta TTCN-3.

La consideración teórica de esta posibilidad, en el sentido de su cumplimineto o no con las definiciones del estándar, la factibilidad práctica de un diseño que cumpliese con la misma y la resolución tecnológica del diseño propuesto son los aspectos principales abordados en la segunda parte de esta tesis.

Organización del documento

El trabajo está organizado de la siguiente forma, en el siguiente capítulo se describirán las características del lenguaje relevantes para el presente trabajo, mientras el resto del trabajo se divide en tres partes, una dedicada al Compilador, la otra a la Solución Dual de interoperabilidad y la última que aborda las conclusiones generales del trabajo realizado.

En la **Parte I** se desarrolla el proceso de construcción del compilador ρTTCN-3, tanto en los aspectos metodológicos y de diseño como respecto su implementación y los resultados obtenidos.

En la **Parte II** se describen los aspectos relacionados a la construcción de la Solución Dual, se describen: el proceso de investigación, su diseño y arquitectura, su integración al compilador ρTTCN-3 de Go4IT y las decisiones tecnológicas adoptadas para su implementación.

Luego nos detendremos en las conclusiones generales del presente trabajo y las perspectivas que presenta la investigación relacionada a TTCN-3 en esta área.

Por último separamos en anexos un conjunto de temas, que si bien nos parecen relevantes, por su extensión no pudimos incluir en el cuerpo del documento, estos son: **Anexo I** detalla la integración del proyecto de construcción ρTTCN-3 en Go4IT, **Anexo 2** contiene el BNF de cobertura de la gramática para la versión A0 de ρTTCN-3, en el **Anexo 3** se brindan detalles de la implementación de ρTTCN-3 y en el **Anexo 4** se brindan detalles de la implementación y despliegue de la Solución Dual.

Capítulo 1

Características del lenguaje TTCN-3

Este capítulo tiene como objetivo brindar una visión general del lenguaje TTCN-3, está pensado para aquellas personas que deban leer este trabajo sin conocer el lenguaje, no siendo necesario su lectura por personas ya familiarizadas con el mismo. La mayor parte de los conceptos brindados aquí están extraidos de los estándares que definen el lenguaje [14, 10, 11, 12, 15, 13] y el libro de introducción al mismo "An Introduction to TTCN-3" [41]

1.1. Definiciones generales de TTCN-3

TTCN-3 es un lenguaje para la especificación de Casos de Prueba (Test Cases). El mismo permite no solo especificar Casos de Prueba simples, sino también, organizar sistemas de pruebas complejos, incluyendo sistemas distribuidos o sistemas en tiempo real.

TTCN-3 puede usarse para la especificación de todo tipo de pruebas de sistemas reactivos, sobre una variedad de puertos de comunicación. Las áreas típicas de aplicación del lenguaje son prueba de protocolos incluyendo comunicación de datos y telefonía móvil, pruebas de servicios, prueba de módulos, prueba de APIs, prueba de componentes basados en CORBA y otros. TTCN-3 no está restringido a pruebas de conformidad, puede usarse para cualquier otro tipo de pruebas, incluyendo interoperabilidad, robustez, regresión, sistemas e integración. [41]

Como lenguaje de programación TTCN-3 recoge las características más relevantes de los lenguajes procedurales modernos, e incluso algunas características menores de los lenguajes orientados a objetos, como la notación con puntos para acceder a las propiedades o métodos de un elemento del lenguaje y en algunas operaciones, pero no de forma consistente en toda la notación del lenguaje.

Además de las estructuras de control tradicionales de estos lenguajes de programación, cuenta con algunas abstracciones especializadas para la realización de pruebas como `TestCases`, `Verdicts`, `AltSteps` y otros.

Si bien de acuerdo a lo especificado en el estándar [14], en la sección **3.1 Definitions**, TTCN-3 no se define como fuertemente tipado, este requiere compatibilidad de tipos en las asignaciones, comparaciones e instanciaciones de acuerdo a lo especificado en el mismo estándar en la sección **6.7 Type Compatibility**.

El lenguaje cuenta con un conjunto de tipos básicos muy amplio y varios mecanismos para la definición de tipos dentro de las que se encuentran `records`, `sets`, `unions`, etc. También en el conjunto de tipos de datos soportado pueden reconocerse especificidades orientadas a las necesidades del testing, soporte para codificaciones de datos como ASN.1, tipos de datos especializados como `bitstring`, `address`, `ports` y `components`. A esto se agregan algunas facilidades como las plantillas (`templates`), para el emparejamiento de patrones (`pattern matching`), así como tipos y valores que ayudan a estas técnicas como `anytype` y `default`.

Mediante la definición de Pruebas Abstractas (Abstract Test Suites - ATS), TTCN-3 logra en gran medida eliminar los detalles de implementación de las definiciones abstractas de las pruebas, consigue esto separando la especificación del comportamiento relacionado con las pruebas del comportamiento relacionado con la plataforma de base, mejorando de esta manera la generalidad y portabilidad de los casos de prueba definidos. De todas formas, esta información debe ser provista a efectos de construir el ejecutable de la Prueba (Executable Test Suite - ETS), el cual se construye para una plataforma concreta. Dependiendo de la plataforma destino Java o C/C++ el ejecutable final será código interpretado o un binario respectivamente, de la misma forma se construirá dependiendo de la plataforma mediante el armado de un jar o la compilación de un binario. Para poder hacer esto el estándar define interfaces que, por un lado, le permiten al desarrollador realizar implementaciones para los adaptadores del SUT e implementar las tareas dependientes de la plataforma de base sobre las que se realizan las pruebas y por otro permiten gestionar la ejecución de los casos de prueba y su trazabilidad.

TTCN-3 incluye las siguientes características esenciales para la realización de pruebas:

- la capacidad de especificar configuraciones de prueba dinámicas y concurrentes
- operaciones de comunicación para mensajes y procedimientos
- habilidad para especificar información codificada y otros atributos
- capacidad de especificar datos y plantillas con poderosos mecanismos de emparejamiento (matching mechanisms)
- parametrización de valores
- asignación y manejo de veredictos
- mecanismos para la parametrización de test suites y selección de test cases
- uso combinado de TTCN-3 con otros lenguajes
- sintaxis, formato de intercambio de datos y semántica estáticas bien definidos
- diferentes formatos de espcificación
- algoritmo de ejecución preciso (semántica operacional)

1.1.1. Entidades de TTCN-3

Un sistema TTCN-3 puede entenderse como un conjunto de entidades que interactuan, donde cada una de las entidades corresponde con cierto aspecto de la funcionalidad en la implementación de un sistema de pruebas. Estas entidades gestionan la ejecución de las pruebas, ejecutando o interpretando el código TTCN-3, comunicándose adecuadamente con el sistema bajo pruebas (System Under Test - SUT), implementan funciones externas y operaciones de temporización.

La figura 1.1 describe la estructura de estas entidades de acuerdo a como la misma está definida en el estándar para TTCN-3. La parte del sistema que trata con la ejecución o interpretación del código TTCN-3 conforma el Ejecutable de Test (Test Executable - TE), estas se corresponden con la ejecución de código producido por un compilador TTCN-3 o mediante un Interprete de TTCN-3. El estándard asume que la implementación del sistema de test incluye la Test Suite Ejecutable (Executable Test Suite - ETS) derivada directamente de la Test Suite Abstracta (Abstract Test Suite - ATS)

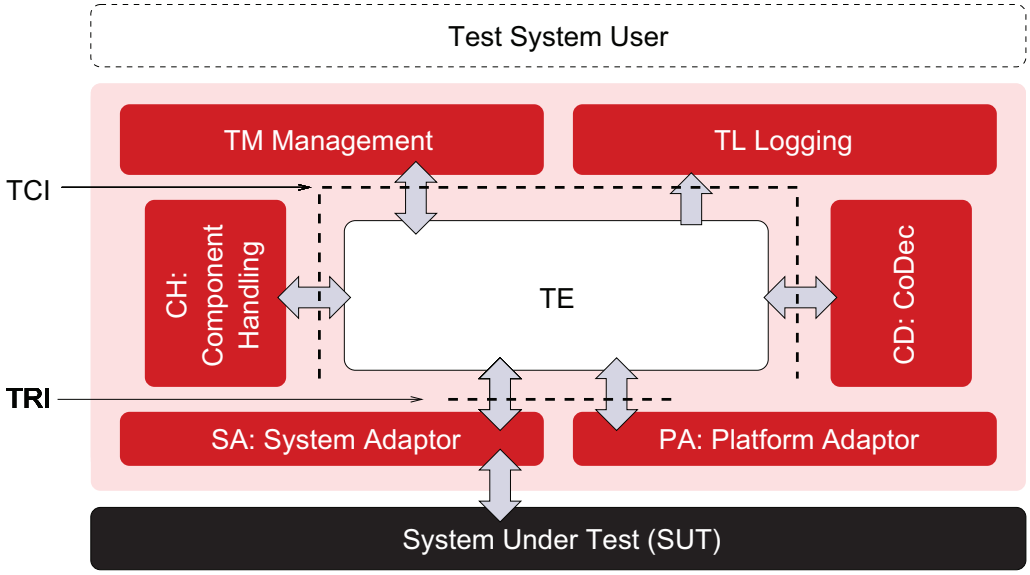

Figura 1.1: Arquitectura de Componentes Estándar

El resto de las entidades del sistema de pruebas TTCN-3 se encargará de manejar los aspectos que no estén especificados en el ATS, los mismos son Gestión de las pruebas (Test Management - TM), Adaptador del Sistema bajo prueba (SUT Adapter - SA) y Adaptador de Plataforma (Platform Adaptor - PA), estos resuelven aspectos como la Interfaz del Usuario, el Control de la Ejecución de las Pruebas, las Trazas de Eventos, así como la Comunicación con el Sistema bajo pruebas y la Plataforma de Base.

Para que estas entidades interactuen con el ETS se definen las Interfaces TRI de tiempo de ejecución (TTCN-3 Run Time Interface) y TCI de control (TTCN-3 Control Interface), las cuales describiremos más adelante.

De esta forma el desarrollodar TTCN-3 especifica el comportamiento del Caso de Prueba (Test Case - TC) en el lenguaje, el cual utilizará, a través de las bibliotecas de su sistema de tiempo de ejecución (Run Time System - RTS), las interfaces para realizar las interacciones necesarias con el sistema bajo prueba y otras funciones desarrolladas en el lenguaje de plataforma. En forma paralela, los desarrolladores de plataforma programan los Adaptadores al sistema (System Adapters - SA) y otras funcionalides necesarias en el lenguaje de base correspondiente, C/C++ o Java, implementando los mapeos de las interfaces necesarias para que se puedan invocar, de acuerdo a las definiciones del estándar desde el RTS. Al momento de la ejecución del caso de prueba el comportamiento del mismo estará dado por el C-ATS, el cual enlaza mediante las interfaces los componentens que se desarrollaron en el lenguaje de plataforma y completan la ejecución del caso.

1.1.2. Adaptador al Sistema bajo pruebas

El Adaptador al Sistema bajo pruebas (System under test Adapter - SA), resuelve la integración entre el sistema de pruebas TTCN-3 y el sistema bajo pruebas, para una plataforma específica de implementación, tanto para la comunicación basada en mensajería como en procedimientos.

El Adaptador está en conocimiento de la relación entre los puertos del componente de prueba (test component) con la interfaz del sistema a probar e implementa esta relación, de acuerdo a la interfaz TRI/SA definida en el estándar [14]. De esta forma el SA es responsable por la propagación

de los mensajes enviados y las operaciones realizadas desde el sistema de pruebas sobre el sistema bajo prueba, también en el otro sentido es responsable por la recepción de cualquier evento de test generado en el SUT y su encolado en las colas del TE. El estándar TTCN-3 define una interfaz entre el SA y el TE, la cual se implementa al programar el Adaptador, de forma que este pueda enviar/recibir los mensajes del SUT al SA y para intercambiar los datos codificados entre las dos entidades en las operaciones de comunicación con el SUT.

1.1.3. Adaptadores de Plataforma

El Adaptador de Plataforma (Platform Adapter - PA), permite la integración de funciones externas a TTCN-3 y también provee al lenguaje con una noción unificada de tiempo basada en la plataforma de base.

Denominamos funciones externas, en este contexto, a funciones implementadas en los lenguajes de plataforma y no en TTCN-3, muchas veces la necesidad de acceder a APIs del Sistema Operativo y de las bibliotecas estándar que este maneja hacen necesario la implementación de este tipo de funciones.

Por otro lado en las actividades de Testing es muy importante la noción de tiempo, los temporizadores son un componente fundamental a la hora de diseñar casos de prueba, ya que estos permiten establecer comportamientos y resultados cuando los sistemas bajo prueba dejan de responder o no lo hacen en los plazos requeridos. Dado que la noción de tiempo computacional esta indisolublemente ligada a la plataforma de base, TTCN-3 prevé mediante el Adaptador de Plataforma la incorporación de ésta a través de los `timers`.

Mediante este Adaptador se accederá a la implementación de todas las funciones externas que se utilizarán en la implementación, así como todos los temporizadores (`timers`) utilizados para sincronizar las diferentes tareas y componentes.

La interfaz TRI/PA, definida en el estándar [14], habilita desde el TE la ejecución de funciones externas, así como el inicio, lectura y parada de los temporizadores, mientras utilizando la parte requerida de la misma interfaz, desde el PA se puede notificar al TE la expiración de los temporizadores.

1.1.4. Gestión de las pruebas

La entidad encargada de la gestión de las pruebas (Test Management - TM) incluye el control de la ejecución y la generación de las trazas de la misma, su funcionalidad puede dividirse de la siguiente forma:

Test Control (TC) es responsable por la gestión general del sistema, una vez que el sistema se inicializa la ejecución de la prueba comienza dentro de la entidad TC, ésta invoca los módulos de TTCN-3, generalmente se encarga de la interfaz de usuario.

Test Logging (TL) es responsable por mantener las trazas del sistema de pruebas.

External CoDecs (ECD) es responsable por codificar y decodificar los datos de los mensajes o procedimientos dentro del TE, existe una interfaz para que estos sean portables entre sistemas.

Component Handler (CH) es responsable por la gestión distribuida de los componentes paralelos, esta distribución puede ser a través de varios sistemas físicos o virtuales.

1.2. Interfaces definidas en el estándar

El lenguaje TTCN-3 logra ofrecer la definición de casos Abstractos de Pruebas (Abstract Test Suites - ATS) mediante la definición de dos interfaces que luego serán implementadas de forma concreta para cada plataforma. De esta forma dado un ATS los detalles de la implementación deben ser provistos, mediante la implementación de dichas interfaces, para poder construir el ejecutable correspondiente (Executable Test Suite - ETS).

Estos detalles se integran dentro del ETS mediante la definición de dos interfaces, la interfaz tiempo de ejecución (TTCN-3 Runtime Interface - TRI) [15] y la Interfaz de Control (TTCN-3 Control Interface - TCI) [16], como veremos más adelante estas interfaces se subdividen de acuerdo a su especialización.

Ambas interfaces se valen de un conjunto de tipos de datos, definidos también en el estándar, que permiten intercambiar la información necesaria entre el ETS, el SUT y la plataforma de base.

Las interfaces son bi-direccionales, en un sentido las partes que invocan a la misma residen en el TE y las operaciones a ejecutar se localizan en las demás entidades, mientras en el otro las invocaciones se realizan desde el SA, la PA o el TCM y las operaciones residen en el TE. las primeras se nombraran como Provistas (Provided), son las operaciones provistas por las demás entidades al TE, mientras las segundas serán las Requeridas (Required), son las operaciones que requieren ejecutar las otras entidades desde el TE. Las interaces se dividen en sub-interfces de acuerdo a la siguiente tabla, donde se muestran además como se distribuyen entre provistas y requeridas.

Interface / Direction	**Provided**	**Required**
TRI triCommunication (TRI-SA)	TE / SA	SA / TE
TRI triCommunication (TRI-SA)	TE / SA	SA / TE
TRI triPlatform (TRI-PA)	TE / PA	PA / TE
TCI Test Management Interface (TCI-TM)	TE / TM	TM / TE
TCI Component Handling Interface (TCI1-CH)	TE / CH	CH / TE
TCI Coding/Decoding Interface (TCI-CD)	TE / CD	CD / TE
TLI Test Logging Interface (TCI-TL)	TE / TL	TL / TE

Tabla 1.1: Tabla de Interfaces

Es así que las diferentes herramientas de TTCN-3 deben proveer un mecanismo para integrar y combinar el ATS con estas implementaciones de las interfaces para construir el ETS.

1.2.1. TTCN-3 Runtime Interface

La TRI define la interacción entre el Ejecutable TTCN-3 (TE), el Adaptor del Sistema (SUT) y el Adaptador a la Plataforma (PA) para una implementación de testing de sistemas. Conceptualmente ésta provee un medio al TE para enviar datos de prueba al sistema bajo pruebas o para manejar los timers, y de forma similar pero en el otro sentido, notificar al TE la recepción de datos de prueba y los timeouts. La TRI está conformada por dos sub-interfaces, la triCommunication y la triPlatform, responsables por la comunicación entre el ETS y el SUT y entre el ETS y e la PA respectivamente. En la siguiente tabla puede apreciarse la relación entre las operaciones de TTCN-3 y la TRI.

TTCN-3 Operation	**TRI Interface**
execute	TriCommunication and TriPlatform
map	TriCommunication
unmap	TriCommunication
send	TriCommunication
call	TriCommunication and TriPlatform
reply	TriCommunication
raise	TriCommunication
action	TriCommunication
start	TriPlatform
stop	TriPlatform
read	TriPlatform
running	TriPlatform
external function	TriPlatform

Tabla 1.2: Tabla de Operaciones de TRI

La interfaz definida para comunicarse con el SUT se usa para enviar y recibir mensajes que son los estímulos para éste y las respuestas respectivas, estos son parte de lo definido en el comportamiento del ATS, esta interfaz es conocida como TRI-SA ya que le permite operar con el Adaptador al Sistema (System Adapter - SA). Dentro de las tareas dependientes de la plataforma se encuentran la implemetación de temporizadores y la ejecución de funciones de bajo nivel provistas por el sistema. Esta interfaz se llama TRI-PA por cumplir el rol de Adaptador de Plataforma (Platform Adapter - PA).

1.2.2. TTCN-3 Control Interface

La TCI se encarga de soportar todas las actividades necesarias para el control de la ejecución de los ETS, como pueden ser iniciar la ejecución y distribución de tareas entre los componentes, trazabilidad, etc. Dentro de la TCI se distinguen TCI-CM para Gestión de Control (Control Management - CM), TCI-CH Manejo de Componentes (Component Handle - CH) y TCI-TL para la gestión de las trazas (Trace & Logging - TL). En la siguiente tabla puede verse la relación entre las operaciones de TTCN-3 y la TCI.

TTCN-3 Operation	**TCI Interface**
send	TCI-CH Provided and TCI-CH Required
call	TCI-CH Provided and TCI-CH Required
reply	TCI-CH Provided and TCI-CH Required
raise	TCI-CH Provided and TCI-CH Required
log	TCI-CH Provided and TCI-CH Required
create	TCI-CH Provided and TCI-CH Required
start	TCI-CH Provided and TCI-CH Required
stop	TCI-CH Provided and TCI-CH Required
kill	TCI-CH Provided and TCI-CH Required
connect	TCI-CH Provided and TCI-CH Required
disconnect	TCI-CH Provided and TCI-CH Required
map	TCI-CH Provided and TCI-CH Required
unmap	TCI-CH Provided and TCI-CH Required
running	TCI-CH Provided and TCI-CH Required
alive	TCI-CH Provided and TCI-CH Required
done	TCI-CH Provided and TCI-CH Required
killed	TCI-CH Provided and TCI-CH Required
mtc	TCI-CH Provided and TCI-CH Required
execute	TCI-CH Provided and TCI-CH Required

Tabla 1.3: Tabla de Operaciones de TCI

Las funcionalidades para el control de ejecución, distribución de tareas, la generación de trazas y otras del estilo son definidas en la Interfaz de control (TTCN-3 Control Interface - TCI), cada una de estas funcionalidades es provista por su propia biblioteca, estas son Gestión de Control (TCI Control Management - TCI/CM), Gestión de Componentes (TCI Component Handle - TCI/CH) y Gestión de Trazas (TCI Trace & Logging - TCI/TL).

Mediante la implementación de estas interfaces, el lenguaje TTCN-3, logra mantenerse abstracto evitando las tareas que están relacionadas con la plataforma. Además de lograr un lenguaje abstracto e independiente de la plataforma, esta estructura del lenguaje permite lograr una eficiente distribución de tareas entre los especialistas en Testing y del lenguaje TTCN-3 con los especialistas en las Plataformas y los sistemas a probar, que se dedicarán específicamente a la construcción de los Adaptadores. Estos niveles de abstracción también permiten el reuso de los ATS específicados en distintas herramientas, proveedores y plataformas.

Finalmente los ejecutables deben construirse en una plataforma concreta, lo cual implica implementar ciertas operaciones utilizando la plataforma de base. TTCN-3 estandariza dos interfaces y mapeos para lenguajes de plataforma, estos son C/C++ y Java.

En general las diferentes herramientas y productos existentes, se basan en una de las plataformas para construir un compilador o interprete de TTCN-3 y brindan ciertas facilidades para la implementación de las interfaces correspondientes en la construcción de soluciones para dicha plataforma. Como introdujimos en a consecuencia de esta situación la industria y la academia han desarrollado esfuerzos disjuntos para crear herramientas en estas plataformas. El resultado es que existen dos familias aisladas de compiladores, herramientas, bibliotecas y soluciones, una en C/C++ y otra en Java. Por esta razón las diferentes comunidades están inhabilitadas de reusar naturalmente las bibliotecas y herramientas desarrollados por la otra cuando se van a crear Casos

de Prueba para los mismos sistemas, o los mismos tipos de sistema. También en algunos casos algunas implementaciones son naturalmente más adecuadas para ser adaptadas usando una de las plataformas, lo cual debería poder realizarse naturalmente. Por ejemplo, los protocolos de red son naturalmente desarrollados usando C/C++, mientras los Web Services son más fáciles y menos costosos de desarrollar en Java. La elección de una única herramienta fuerza a los desarrolladores a pagar sobre-costos para cruzar de plataforma cuando se dan estos casos o implementar la adaptación en una plataforma menos adecuada a la naturaleza del problema.

1.3. Abstracciones del lenguaje especializadas en Testing

La estructura principal de definiciones de TTCN-3 son los módulos, estos definen y nombran un bloque de ejecución, dentro del mismo hay una parte de definiciones y otra de control, el módulo es capaz de recibir parámetros. En la parte de definiciones pueden realizarse todas las declaraciones necesarias como `test components`, `ports`, `types`, `templates`, `test cases`, `functions`, etc. En la parte de control se realiza las llamadas a los `test cases` y se controla su ejecución.

Algunas de los conceptos especializados en Testing que incorpora el lenguaje son interesantes de conocer, a continuación los desarrollaremos brevemente pues entendemos que ayudarán comprender como opera el lenguaje.

Test Components El tipo Componente (Component Type) permite definir los puertos, asociados con el mismo, que se utilizaran para comunicarse con otros componentes y con el sistema bajo pruebas, la definición de los mismos se realiza en la parte de definiciones de cada módulo y es local, dentro del componente también pueden declarase variables y timers, estas definiciones son visibles desde cualquier Testcase que utilice el componente, lo cual se explicita mediante las palabras clave ***runs on***. De acuerdo al rol que cumplen, podemos distinguir tres clases o tipos diferentes de componentes:

1. Abstract Test System, interfaz con el SUT definida como un componente
2. Main Test Component (MTC), componente principal donde se iniciará por defecto la ejecución del TC.
3. Parallel Test Component (PTC), cero o más componentes para la ejecución concurrente de actividades que se crean explícitamente durante la ejecución del ETS.

TTCN-3 permite la configuración dinámica de tests concurrentes, esta configuración consiste en la especificación de un conjunto de componentes interconectados por sus puertos correctamente definidos y una interfaz con el sistema de pruebas (test system interface), que define claramente los bordes con el sistema a probar.

En cada configuración existirá un solo componente del tipo MTC, el resto de los componentes serán del tipo PTC. Mientras el MTC es creado automáticamente por el sistema cuando comienza la ejecución de cada Testcase, desde el cuerpo de los distintos componentes se pueden crear dinámicamente nuevos componentes del tipo PTC con la sentencia `create`, de acuerdo al comportamiento especificado en el cuerpo de los mismos.

En la figura 1.2 se describe la configuración típica de los componentes en un sistema de pruebas.

Test Cases. Los Test Cases son la abstracción básica donde se especifica el comportamiento de cada caso de prueba, cada TC se ejecuta en el contexto de un componente de tipo MTC que es asociado al momento de la definición del mismo. Los Test Cases son ejecutados, desde la parte de

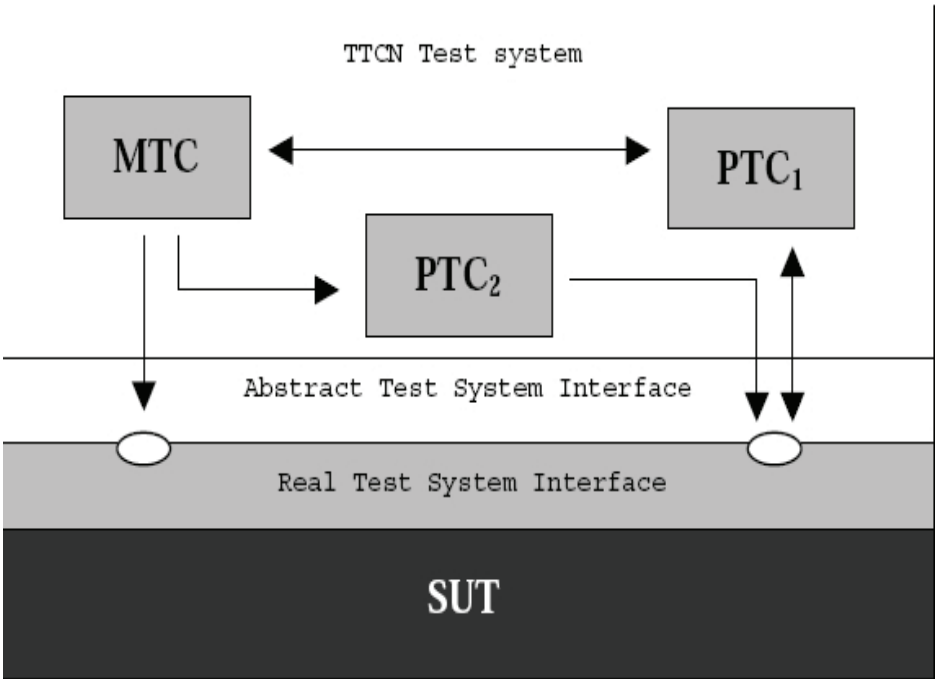

Figura 1.2: Configuración Típica de Componentes

Control de los Módulos, mediante la sentencia ***execute***. Desde el punto de vista del Lenguaje los Test Cases son una clase especial de funciones que siempre retornan un valor del tipo `verdicttype`, desde el punto de vista de su contenido un Testcase puede incluir cualquier sentencia que puede incluirse en una función.

Cuando un Testcase es invocado el MTC correspondiente es creado, los puertos de éste y de la Interfaz con el Sistema de Pruebas son instanciados y el comportamiento del Testcase es ejecutado en el MTC, para lograr que estas operaciones ocurran de forma implícita el Test Case Header incluye:

1. interface(obligatorio) con la palabra clave `runs on` se referencia el tipo del del MTC y deja disponibles los puertos dentro de este.

2. test system(opcional) con la palabra clave `system` se referencia el tipo que define la interfaz con el sistema de pruebas.

Cuando solo se instancia el MTC, puede omitirse el componente para el Test System y las definiciones para este son tomadas del MTC.

Verdicts. `verdict` es un tipo de datos que expresa el resultado de un caso de prueba (`Testcase`), el mismo puede tener los siguientes valores: `pass, inconc, fail, none, y error`. Las operaciones permitidas sobre el las variables del tipo `verdict` son solo para obtener `getveredict()` o establecer `setveredict()` su valor, estas operaciones solo pueden ser usadas en `Testcases`, `Altsteps` o `Functions`. Los cambios al asignar veredictos deben seguir las reglas establecidos en la siguiente tabla. No todos los valores de verdicts pueden ser asignados con la operación set, el valor error es establecido por el sistema en caso de que ocurra un error en tiempo de ejecución, este valor no puede ser sobreescrito por ningún otro no puede ser devuelto por la operación `getveredict()`.

	Nuevo valor asignado			
Valor actual	**pass**	**inconc**	**fail**	**none**
none	pass	inconc	fail	none
pass	pass	inconc	fail	pass
inconc	inconc	inconc	fail	inconc
fail	fail	fail	fail	fail

Tabla 1.4: Resultados de setveredict()

Cada componente mantendrá su propio veredicto local, este es creado al momento de la creación del componente y mantendrá la traza del veredicto del componente mientras éste este activo. Adicionalmente existe un veredicto global del Test Case que se actualizará automáticamente cada vez que un componente termina su ejecución, este veredicto no puede ser actualizado explícitamente.

Abstract Test System Interface. En una solución real de testing los casos de pruebas deben comunicarse con el SUT, pero la especificación real física de estas conexiones no es parte de TTCN-3, en cambio este soporta la especificación de una interfaz abstracta para cada caso de prueba (TC), esta es como la definción de un componente, donde se declara una lista de los puertos que podrán ser usados para comunicarse con el SUT. Si bien esta lista es estática las conexiones definidas en ellas podrán ser relacionadas dinámicamente en tiempo de ejecución con los puertos físicos, brindados por el SUT mediante la implementación de las interfaces correspondientes, mediante las operaciones de `map` y `unmap`.

Communication Ports. Los puertos son una de las abstracciones clave en el lenguaje TTCN-3, relacionada con sus capacidades de Testing, estos son el medio de comunicación entre los componentes del sistema de pruebas y también son el medio para comunicarse con el sistema bajo pruebas a través de la `Test System Interface`.

Los puertos son definidos siempre en la parte de declaraciones de un componente, cada puerto se modela como una cola FIFO para la gestión de los mensajes entrantes o las llamadas a procedimiento mientras las procesa el componente correspondiente. Los puertos pueden ser de entrada (`in`), salida (`out`) o mixtos (`inout`).

Los puertos pueden ser basados en mensajes, procedimientos o mixtos determinando el tipo de operaciones que se pueden realizar sobre ellos, las operaciones en cada caso son:

- **Message based**
  - `send`, Send message
  - `receive`, Receive message
  - `trigger`, Trigger on message
- **Procedure based**
  - `call`, Invoke procedure call
  - `getcall`, Accept procedure call from remote entity
  - `reply`, Reply to procedure call from remote entity
  - `raise`, Raise exception (to an accepted call)

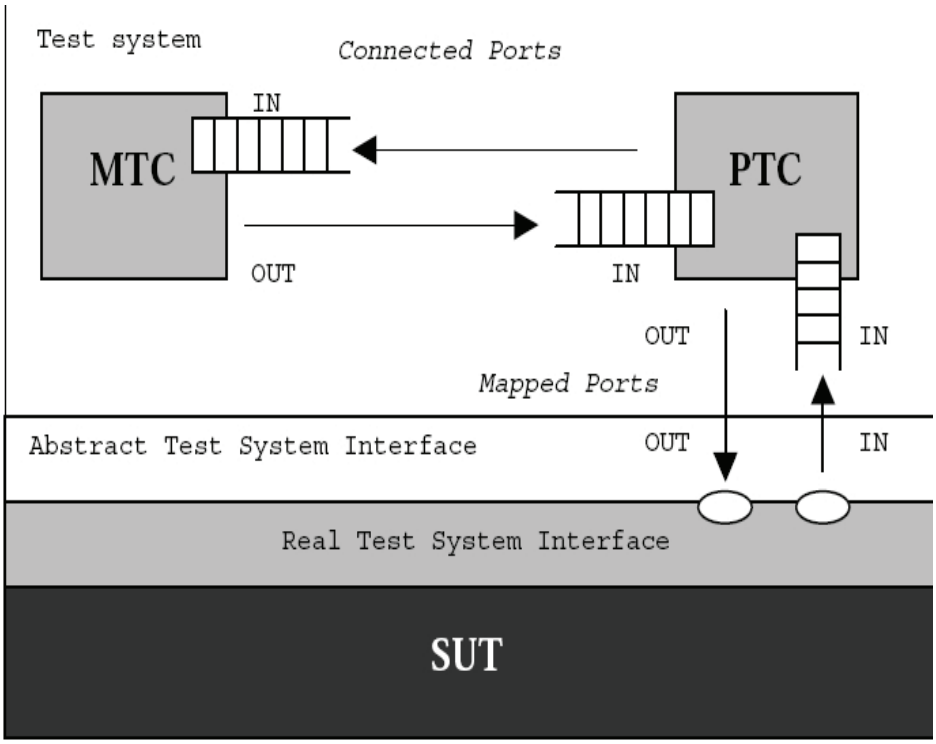

Figura 1.3: Puertos, conexiones y mapeos entre componentes y el SUT

- `getreply`, Handle response from a previous call
- `catch`, Catch exception (from called entity)

Independientemente del tipo de comunicación del puerto las operaciones de estos pueden agruparse en las de envío -`send`, `copy`, `reply` y `raise`- y las de recepción -`receive`, `trigger`, `getcall`, `getreply`, `catch`.

Alt Statement and Altsteps. La sentencia `alt statement` hace posible especificar un conjunto de sentencias alternativas, que llamaremos ramas. Las ramas se ejecutaran en función del estado del sistema de testing en el momento de su evaluación, mediante el ***snapshot*** se brinda un mecanismo para evaluar en un momento dado los diferentes eventos del sistema de forma asincrónica, permitiendo manejar eventos de comunicación, temporizadores, finalización de componentes y otros. Cada rama tiene además una condición de entrada que será evaluada previamente al chequeo del evento correspondiente, mientras esta condición no se cumpla la sentencia no se detendrá en dicha rama.

Mediante la especificación de las operaciones correspondiente como `receive`, `trigger`, `getcall`, `getreply`, `check`, `timeout`, `done` or `killed` cada rama del `alt statment` estará chequeando por la ocurrencia de estos eventos. Para resolver la evaluación de los eventos y otros valores en cada rama se cuenta con el concepto de ***snapshot***, este es un estado parcial del `test component`, que contiene todo lo necesario para la evaluación de la sentencia, y es tomado al instante en que comienza a ejecutarse el `alt statment`.

Los `altsteps` sirven para especificar comportamientos por defecto o para estructurar las alternativas del `alt statement`, los `altsteps` manejan definiciones de alcance similares a las de

una función, permiten el pasaje de parámetros y la definición de variables locales. El cuerpo de un `altstep` está constituido por un conjunto de alternativas, llamadas ***top alternatives***, las reglas sintácticas de las ***top alternatives*** son las mismas que las de las alternativas de un `alt statement`. El `altstep` puede utilizar operaciones sobre puertos y temporizadores, para esto puede contar con la clausula `runs on`, la cual deja disponibles los puertos y timers del componente dentro del `altstep` o puede también recibirlos como parámetros.

La llamada de un `altstep` estará siempre relacionada a una sentencia `alt statement`, ya sea como una opción declarada por defecto o como una llamada explícita en una de sus ramas.

Templates y Pattern Matching. Los `templates` son estructuras de datos especiales para testing, se usan tanto para transmitir un conjunto de valores o para chequear cuando un conjunto de valores recibidos coincide con la especificación de estos. Una variable `template` puede contener además de valores mecanismos de chequeo de coincidencias de patrones llamado ***pattern matching***.

Los mecanismos de ***pattern matching*** pueden agruparse de la siguiente forma:

- **Valores específicos**
  - `expression,` una expresión que evalua a un valor
  - `omit,` valor omitido
- **Símbolos especiales que pueden usarse en lugar de valores**
  - `(...),` lista de valores
  - `complement (...),` complemento de lista de valores
  - `?,` comodín para cualquier valor
  - `*,` comodín para cualquier valor o vacío
  - `(inf .. sup),` un rango de números enteros o flotantes
  - `superset,` por lo menos todos los elementos listados o más
  - `subset,` como máximo todos los elementos listados o menos
- **Símbolos especiales que pueden ser usados dentro de un valor**
  - `?,` comodín para un elemento en los tipos string, array, record o set of
  - `*,` comodín para cualquier cantidad de elementos consecutivos en los tipos string, array, record o ningún elemento
  - `permutation,` cualquiera de los elementos de la lista en cualquier orden, estos podrían incluir ? o *
- **Símbolos especiales que describen atributos de los valores**
  - `length,` restricciones para el largo de un valor string, la cantidad de elementos de record of, set of y arrays
  - `ifpresent,` para chequear por valores en campos opcionales, aplica cuando estos no están omitidos

Nuestra impresión general, una vez que estudiamos el lenguaje y vimos algunos ejemplos de implementaciones sencillas con este, es que efectivamente es un lenguaje completo, que desde el punto de vista general cuenta con las herramientas necesarias para programar y desde el punto de vista del testing ofrece un conjunto de abstracciones útiles para los desarrolladores de pruebas.

Dentro de los aspectos no técnicos más importantes destacamos el hecho de que el mismo esté estandarizado y cuente con el apoyo de organizaciones como ETSI, lo cual, además del propio esfuerzo que destina ETSI para mantenerlo, le da un impulso importante tanto en la academia como en la industría, especialmente en Europa y sus áreas de influencia.

En cuanto a sus capacidades, la posibilidad de configurar de forma dinámica pruebas concurrentes, los diferentes mecanismos de comunicación que soporta, algunos elementos que lo hacen aplicable en áreas específicas de la industria como el soporte a ASN.1, lo presentan como una alternativa muy interesante en varias áreas de aplicación.

Además de estas capacidades tiene atributos importantes en cualquier lenguaje como modularidad, extensibilidad mediante atributos y el uso de las interfaces definidas por el estándar, así como una clara y completa definición sintáctica, semántica y operacional.

Todos estos aspectos relacionados lo presentan como una herramienta con un gran potencial para extender su uso a nuevas áreas de la industria, así como otros tipos de pruebas.

Parte I

Un compilador abierto para TTCN-3: μTTCN-3 & ρTTCN-3

Capítulo 2

Objetivos y Alcance del Compilador

Para poder realizar los experimentos vinculados a nuestras líneas de investigación en TTCN-3, entendimos necesario disponer de un compilador, no solamente para su uso con menos restricciones, sino también para su modificación y extensión de acuerdo a las propuestas de cambios que pudiesen surgir. Por esta razón el desarrollo de un compilador experimental para el lenguaje TTCN-3 se constituyó en parte de las tareas de este trabajo de tesis. Complementariamente, la construcción del compilador nos aportaría un conocimiento fundamental sobre el lenguaje TTCN-3 para las siguientes etapas de nuestro trabajo.

2.1. Alcance Inicial

En primera instancia definimos realizar la implementación del compilador para un subconjunto del lenguaje TTCN-3, que nombramos μTTCN-3, con el objetivo de simplificar el proceso de desarrollo y tener un compilador operativo en el menor tiempo posible, con los escasos recursos disponibles. El criterio utilizado para la determinación de este alcance inicial fue que el subconjunto definido del lenguaje debía permitir la implmentación de Test Suites reales, de la complejidad suficiente para las necesidades de nuestro trabajo posterior. De esta forma, el compilador implementado proveería una base adecuada para la experimentación en las áreas de interés del lenguaje definidas por nosotros y también con ciertas tareas definidas por el equipo de ARMOR [4] en cuyo contexto estábamos realizando estas tareas de forma colaborativa. En este sentido a efectos de poder experimentar en el área de interoperabilidad se requería disponer de las primitivas para el manejo de puertos y las funcionalidades relacionadas a la TRI, las cuales de todas formas son parte básica del lenguaje.

Más allá de nuestra intención inicial, las interrelaciones entre las diferentes construcciones del lenguaje, volvieron difícil seleccionar los lugares donde realizar un corte en la implementación de la gramática logrando que esta fuese operativa, por otro lado avanzamos en la automatización del pasaje del BNF del lenguaje a la especificación del analizador sintáctico, de esta forma resultó más sencillo especificar casi completamente el analizador para la gramática, dejando implementaciones automáticas de la semántica asociada en aquellas partes donde no se requería que la implementación del árbol de análisis sintáctico quedase completa.

2.2. Alcance final - Integración con Go4IT

Los avances realizados en este sentido fueron de interés para el consorcio Go4IT [22], contábamos ya con un parser operativo aunque aún limitado en las producciones semánticas generadas y con

mínimos avances en las otras etapas del compilador, coincidió entonces con las necesidades del proyecto Go4IT en su Package 2, el cual definió en ese momento la necesidad de desarrollar un compilador abierto de TTCN-3 y visualizó en nuestro trabajo la oportunidad de partir de una solución más avanzada.

El objetivo de Go4IT en relación al compilador de TTCN-3 se alineaba en algunos aspectos con el nuestro. El proyecto buscaba poner a disposición de la comunidad una herramienta abierta y libre TTCN-3, que le permitiera promover sus objetivos, en el área de pruebas de conformidad para IPv6, con independencia de las versiones del producto disponibles en el mercado. Esto le requería no solamente una versión operativa sino más completa, del compilador y las herramientas asociadas con éste, de la que nosotros nos habíamos planteado originalmente.

La posibilidad que se planteó en este contexto fue incorporarnos a un equipo mayor, conformado a su vez por equipos de varios países, que aportarían trabajo, conocimientos y experiencia en diferentes áreas de la construcción de compiladores y del lenguaje TTCN-3, esto tendría como contrapartida la ampliación del alcance planteado originalmente. Evaluando la relación costo/beneficio planteada y nuestras posibilidades reales, concluimos que la mejor opción era incorporarnos al proyecto, ya que la extensión requerida a la etapa de Análisis no era mayor que los aportes que el proyecto brindaría como contrapartida, en las etapas de Traducción, construcción del Sistema de Tiempo de Ejecución (RTS) y especialmente el proceso de pruebas posterior, considerando además como un elemento positivo de por si la extensión del alcance propuesto.

En la figura 2.1 se describe la arquitectura inicial del compilador ρTTCN-3, que quedó definida ya en el contexto del proyecto de Go4IT.

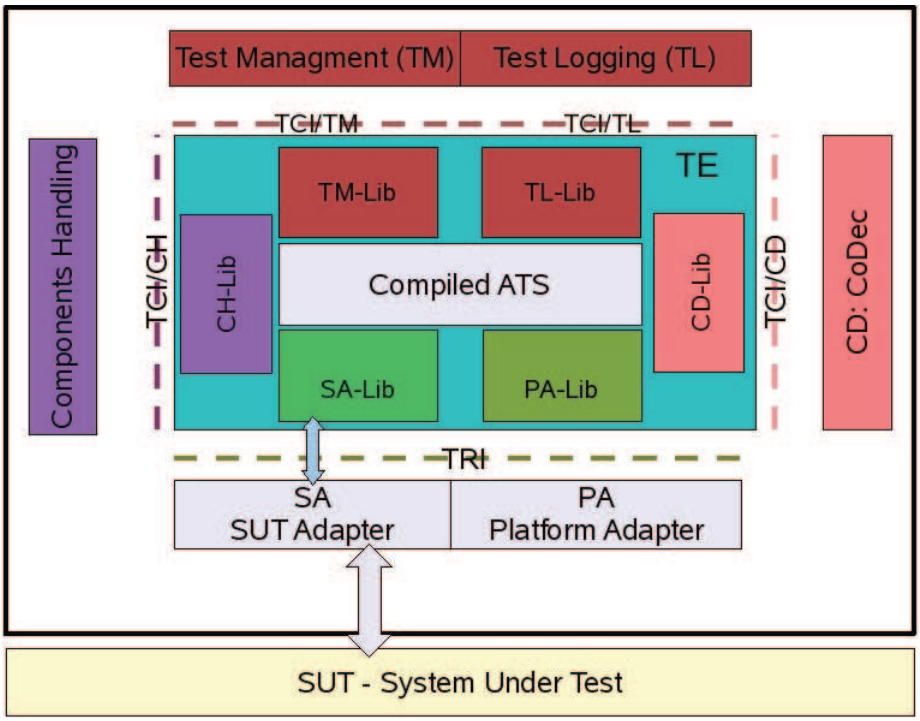

Figura 2.1: Go4IT ρTTCN-3 Arquitectura inicial del compilador

Capítulo 3

Investigación preliminar para la construcción del compilador

Con el objetivo de disponer del compilador μTTCN-3 para conocer y experimentar con el lenguaje, nos propusimos algunas actividades de estudio e investigación para determinar la metodología y herramientas a utilizar en la construcción del mismo así como definir más detalladamente el alcance que daríamos a este.

Dados los cometidos específicos del desarrollo del compilador y los recursos limitados con los que contábamos inicialmente se analizó la posibilidad de definir un alcance para la primera etapa lo más reducido posible. Para la definición de este alcance inicial, al no contar con experiencia previa en TTCN-3, nos planteamos la necesidad de realizar actividades que nos ayudasen conocer el lenguaje, con el objetivo básico de construir el compilador μTTCN-3 y de lograr una versión operativa de acuerdo a nuestras necesidades de experimentación.

En primer término nos introdujimos en el lenguaje, luego profundizamos estos conocimientos estudiando la definción formal del lenguaje, especialmente la gramática y en algunos temas particulares la semántica operacional.

El examen de algunos ejemplos nos ayudo mucho a completar la comprensión de los mecanismos usados por el lenguaje, lamentablemente en esta primera etapa por algunas dificultades formales y prácticas no logramos contar con un ambiente para experimentar con el lenguaje, situación que reafirmaba la necesidad de disponer de una plataforma más accesible para la investigación en torno al lenguaje.

A partir de este relevamiento inicial, requerimos la opinión del equipo de desarrolladores TTCN-3 de ARMOR [4]/IRISA [27], que utilizaba el lenguaje para las pruebas de conformidad en el contexto de Go4IT, a efectos de contar con un punto de vista práctico sobre los elementos del lenguaje realmente utilizados por ellos y aquellas carácteristicas del mismo que no utilizaban usualmente.

En paralelo realizamos un relevamiento sobre experiencias actuales en el desarrollo de compiladores para lenguajes de programación, tecnologías y herramientas disponibles, con el fin de evaluar el camino a seguir en la construcción del compilador ρTTCN-3.

Para evaluar y realizar una síntesis de todos estos insumos, mantuvimos una serie de entrevistas con referentes de Lenguajes y Compiladores del INCO. Basados en los elementos y opiniones recabados tomamos algunas decisiones relacionadas con el alcance inicial del compilador y también respecto a la metodología para el desarrollo del mismo que orientaron nuestro trabajo posterior.

3.1. Estudio del Lenguaje

Para un primer acercamiento con el lenguaje tomamos como base el libro de introducción al mismo, "An Introduction to TTCN-3" [41], en el cual se desarrolla un acercamiento práctico al lenguaje basado en ejemplos. Este libro nos permitió entender el lenguaje tanto desde un punto de vista tradicional, incluyendo su sintaxis, tipos de datos y estructuras de control, así como sus abstracciones originales como lenguaje dedicado al testing, por ejemplo las estructuras de control ALT STEPS o los conceptos de PUERTOS para comunicarse con los sistemas bajo prueba, las enseañzas más relevantes de este proceso son expuestas en el Capítulo 1.

En una etapa posterior, dispusimos del compilador y sus herramientas relacionadas *TTworkbench* de la empresa *Testingtech* [39] para experimentar con algunas de las prácticas propuestas en el libro y algunos otros casos de prueba provistos por ARMOR, como los relacionadas con IPv6. De esta forma logramos entender el lenguaje desde la perspectiva del usuario desarrollador de TTCN-3, pudimos así valorar las abstracciones especializadas para la realización de pruebas provistas por el lenguaje y su riqueza en otros aspectos como la cobertura de tipos y los mecanismos de conversión disponibles para serializarlos e intercambiarlos con el SUT en cada caso. También, desde la perspectiva del desarrollo de los adaptadores, entendimos las dificultades y el trabajo extra necesario para llegar desde el ATS al ETS, elementos que nos habían sido trasmitidos previamente por los integrantes del equipo de ARMOR/IRISA y que fundamentan algunas de las investigaciones que nos propusimos en el área de interoperabilidad.

Más allá de conocer y comprender el lenguaje, primera etapa imprescindible para poder seguir avanzando en nuestras tareas, nos enfocamos luego en identificar algún tipo de métrica que nos pudiese ayudar a dimensionar, al menos de forma relativa, el esfuerzo que implicaría dar soporte a cada una de las abstracciones y los diferentes elementos que componen el lenguaje TTCN-3. Esta necesidad surgía de uno de nuestros objetivos en esta etapa, definir el alcance de μTTCN-3.

Trabajando en este sentido fue que nos dispusimos a evaluar la complejidad de los diferentes elementos del lenguaje como: los tipos datos soportados por este, los mecanismos de abstracción del mismo, incluyendo las definiciones de funciones, procedimientos, test cases, recursividad y anidamiento en las definiciones, etc. Para poder profundizar estos aspectos debimos recurrir al estudio del BNF, provisto en la definición del estándar, para el lenguaje TTCN-3 en el Anexo A.1.6 ***TTCN-3 syntax BNF productions*** del documento ***TTCN-3 Core Language*** [14], como forma de lograr analizar todas las alternativas que podían generarse para las diferentes construcciones del lenguaje, en algunos casos debimos recurrir aún a más detalles sobre el comportamiento esperado de las sentencias a efectos de entender su funcionamiento y grado de complejidad, para esto recurrimos a la especificación de la semántica operacional en el documento ***TTCN-3 Operational Semantics*** [12].

3.2. Datos empíricos sobre el uso del lenguaje

Lo novedoso del lenguaje en su versión 3, con la consecuente falta de información empirica en torno al mismo, y las dificultades, ya mencionadas en la sección anterior, para disponer de una plataforma de experimentación, fueron una de nuestras preocupaciones a lo largo del proyecto, especialmente a la hora de tomar las definiciones inciales del mismo. Estos factores se manifestaban no tanto en la construcción del compilador y su analizador, que responden a la especificación formal del lenguaje, disponible en forma de BNF en el estándar [14], sino al momento de tomar decisiones sobre el alcance o las necesidades concretas de la herramienta que resultaría como producto de nuestro trabajo.

Para atacar esta necesidad de información nuestra fuente de referencia empírica fue el grupo ARMOR/IRISA, a la cual tuvimos acceso gracias a nuestra interacción con algunos de sus integrantes en el contexto del proyecto Go4IT En particular tuvimos la oportunidad de interactuar directamente con ellos en una visita que realizamos a RENNES 1 durante ese período, preparatoria de las jornadas de trabajo en Beijing. El grupo contaba con un equipo con vasta trayectoria en el desarrollo de casos de prueba (Test Cases) utilizando TTCN-3, especialmente en el área de testing de protocolos de red. Este equipo estaba trabajando en el desarrollo de pruebas de conformidad en IPv6 en el contexto del proyecto Go4IT, desarrollando en TTCN-3 un paquete para automatizar las pruebas de conformidad de dispositivos para esta nueva versión del protocolo IP.

El paquete construido, el*IPv6 Testing Toolkit* [34], resuelve de forma flexible alguna de las tareas más costosas que requería TTCN-3 para abordar este tipo de pruebas, podemos resaltar aquí la automatización de la generación de CoDecs y la publicación de un framework para la ejecución de funciones externas y el uso de Adaptadores. El trabajo realizado dejó en el equipo una experiencia importante y un conocimiento profundo de las áreas del lenguaje relacionadas, los cuales pudimos capitalizar en beneficio de nuestro proyecto. Si bien somos conscientes que esta información era parcial, ya que respondía a las necesidades de un área específica de aplicación de TTCN-3, pruebas de conformidad de dispositivos para protocolos de red, aún así la experiencia era igualmente válida para nuestros objetivos concretos.

Uno de los aspectos principales que nos permitió definir la experiencia de los especialistas, junto a algunas métricas generadas a partir de su trabajo, fue la validación en el aspecto funcional, que era posible construir casos de prueba de interés para la industria, usando solo un subconjunto del lenguaje.

Tipo	**Ocurrencias**
Universal string	0
Hexstring	18
Bitstring	134
Octetstring	183
Charstring	267
Set	34
Union	75
Record	285

Tabla 3.1: Uso de tipos de datos - IPv6 ToolKit

En particular los juegos de pruebas para conformidad para IPv6 han sido implementados utilizando solo un subconjunto de las características de TTCN-3, quedando fuera algunas de estas que significarían una diferencia sustancial en el esfuerzo de desarrollo de nuestro compilador. Un caso relativamente sencillo de medir es el uso de los diferentes tipos de datos, para esto tuvimos en cuenta las ocurrencias de los mismos en el código fuente del paquete desarrollado para IPv6 pudimos observar que algunos de ellos, como el `Universal string`, no fueron usados en ningún caso y otros solo en algún caso aislado. Otro ejemplo es la utilización de un único hilo de control dejando de lado el uso de componentes paralelos (Paralel Components), esto se debe a que por la naturaleza de los protocolos de red y las pruebas sobre los mismos es sencillo modelar un componente del sistema con un único hilo, ya que en las pruebas de conformidad se estimula y espera la respuesta al estímulo de forma serializada, para luego proceder con el siguiente estímulo. Tampoco fueron utilizadas en estas implementaciones las definiciones anidadas y otras características del lenguaje

que seguramente responden a necesidades específicas de otras áreas de aplicación o a aplicaciones más complejas.

En definitiva la información recogida del código fuente generado en Go4IT para las pruebas de conformidad de IPv6 fueron nuestro principal insumo para la generación de métricas, las cuales nos permitieron validar ciertos limites en el alcance propuesto a nuestra versión del compilador.

3.3. Alternativas evaluadas para la construcción del compilador

Contábamos inicialmente con un conocimiento académico básico sobre la construcción de compiladores, la cual nos permitía realizar una implementación de acuerdo a las pautas establecidas por estos. Para validar este camino decidimos evaluar las nuevas alternativas tecnológicas existentes para abordar el desarrollo del compilador, en la búsqueda especialmente de disminuir los tiempos del proyecto mediante el uso de nuevas herramientas. Para esto recurrimos a algunos libros de referencia en este tema, en particular optamos por el clásico "Compilers: Principles, Techniques, and Tools." [5] y su segunda versión [6], también realizamos un relevamiento en Internet donde buscamos información sobre proyectos de automatización en la construcción de compiladores, con el objetivo de determinar si se disponía de algún nivel más de asistencia ampliamente aceptado en la industria para la construcción de compiladores. Seleccionamos algunas de las propuestas recientes para el desarrollo de compiladores con diferentes tipos de asistencias automátizadas, entre ellas decidimos profundizar en algunas como el BNF Converter [2] y Cactus [1] que se enfocaban en la parte del trabajo que entendíamos podía admitir mayores niveles de automatización. Como conclusión general podemos resumir que estas herramientas ofrecen algún paso extra de automatización a las tradicionales de yacc/bison y lex/flex, permitiendo partir de algunas especificaciones en BNF anotadas y EBNF o especificaciones equivalentes, para generar automáticamente las especificaciones de los analizadores sintácticos y léxicos correspondientes. Como contrapartida las mismas generan ciertos niveles de dependencia y algunas presentan dificultades a la hora de cambiar la plataforma destino del compilador, por ejemplo pasar de compilar C/C++ a Java o viceversa.

3.4. Opciones realizadas

Tomando en cuenta las características del lenguaje, las métricas y experiencia mencionadas, nuestras condiciones particulares para el desarrollo y las alternativas de implementación existentes, en conjunto con la opinión de nuestros referentes en el área de compiladores de InCo/UdelaR, concluímos en el plano más general, que una metodología tradicional en el desarrollo del compilador nos brindaría mayor flexibilidad a la hora de abordar cambios en el proyecto, el cual desde su planteo inicial estaba naturalmente muy expuesto a estos. Esta definición, que tiene consecuencias directas en el diseño del compilador como veremos en la sección 4 y también en la metodología de desarrollo, significó la división del compilador en módulos y etapas en su desarrollo. Esta situación facilitó a *posteriori* la posibilidad de trabajar en equipos independientes en cada uno de estas etapas o módulos, a pesar de que esta no era una restricción en el inicio del proyecto.

Con respecto a las herramientas de desarrollo, en lugar de comprometernos con alguna de ellas en particular, preferimos considerar algunos de los conceptos que estas proponían en nuestro propio proceso de desarrollo, de esta forma nos quedamos con los aspectos que nos convenían y descartamos otros que a para los objetivos del proyecto Go4IT no nos parecián convenientes. Esto se traduciría posteriormente en el desarrollo de algunas herramientas o utilidades que nos facilitaron las iteraciones en el desarrollo del módulo de análisis del compilador y el trabajo integrado con los demas equipos del proyecto.

En lo que tiene que ver con acotar el alcance del compilador, uno de los factores más importantes considerados fue la definición de implementar una única línea de control para las pruebas, excluyendo así del mismo la capacidad de implementar Casos de Prueba Paralelos (Parallel Test Cases - PTC). Si bien el estándar definido por ETSI para TTCN-3 prevé y promueve, la existencia de múltiples hilos de control con uno principal responsable de la coordinación, mediante la experiencia del grupo ARMOR/IRISA en el uso de TTCN-3 para el testing de RIPng [35], se pudo validar que la utilización de un único hilo de control simplifica el proceso de desarrollo, impactando positivamente en el procesos de creación de las Test Suites y en el proceso posterior de ejecución de las mismas al simplificar el despliegue y ejecución de los mismos.

Basados en la experiencia de ARMOR/IRISA, relacionada a las pruebas de conformidad de IPv6, los PTCs pueden ser sustituidos por varios casos de prueba simples. La razón para esto es que en este tipo de pruebas los eventos se serializan y envían al SUT cuando este está estable, esperando para el envío del siguiente estímulo a que vuelva a estabilizarse, esta situación refuerza la posibilidad de prescindir de los PTCs en estos casos. Por otro lado, dar soporte a los PTCs aparece como una de las características más costosas y complejas a ser desarrolladas en el compilador, especialmente en el módulo del sistema de tiempo de ejecución dar este soporte tendría implicancias en la arquitectura y tecnología a usar.

Si bien esta metodología no siempre es aplicable, en muchos casos de test de conformidad, cuando se evalua el comportamiento de un dispositivo ante un estímulo concreto de forma serializada, se encontró que la misma puede ser suficiente y más simple de implementar y usar.

Esta característica diferenciadora podría, además de simplificar la tarea de desarrollo del compilador, servir para contrastar dos metodologías de implementación, al forzar la utilización de un hilo único de control en contraste al uso de componentes paralelos tal como promueve el estándar, un tema interesante a evaluar especialmente en el área de conformidad donde TTCN-3 tiene uno de sus mayores campos de aplicación.

Una de las características de TTCN-3 es contar con un potente y variado conjunto de datos, el cual está pensado para que el lenguaje pueda ser utilizado de forma amplia, pero claramente no todos estos tipos de datos son utilizados en los diferentes nichos de aplicación de TTCN-3. En este sentido se decidió limitar la cobertura de los tipos de datos a aquellas estadísticamente más utilizadas en ARMOR, basados en las métricas oportunamente obtenidas de la experiencia del grupo en el proyecto Go4IT. Si bien existen algunos tipos de datos que, por su nivel de estructuración, parecen requerir un esfuerzo especial para su inclusión, el solo hecho de reducir en apróximadamente un 30 % los tipos de datos a soportar tendría definitivamente un impácto directo en los tiempos del proyecto.

Por último y en virtud de la relación costo beneficio que éste ofrecía y su relativo poco uso, excluimos del alcance las definiciones anidadas, las mismas parecen ofrecer niveles de dificultad superiores tanto a nivel del módulo de análisis como del sistema de tiempo de ejecución.

Capítulo 4

Metodología, Diseño y Tecnología

Nuestra decisión de usar una metodología tradicional para la construcción del compilador nos llevó a un desarrollo orientado por la sintaxis, con una división del trabajo y el propio compilador en varias etapas o módulos, de acuerdo a lo recomendado por dicha metodología y a las características particulares de nuestro desarrollo. Una descripción detallada sobre estos tópicos puede encontrarse en el libro de referencia de A .Aho, R. Sethi, and J. D. Ullman [5].

Desde una perspectiva tecnológica, siguiendo las recomendaciones recibidas y nuestra propia experiencia, seleccionamos como herramientas de análisis lexicográfico y sintáctico Flex y Bison, ya que estas se mantienen como un estándar de facto en la industria y facilitan además la migración o alternancias entre C/C++ y Java como posibles plataformas generación.

Como plataforma de base para el desarrollo de esta versión del compilador se seleccionó C/C++, con el objetivo implícito de trabajar con el paradigma de objetos, esta elección coincidía con la plataforma inicial de destino para la traducción y es además la plataforma mayormente usada en ARMOR/IRISA/ETSI.

Para complementar la herramienta de desarrollo C/C++ bajo el paradigma de objetos se evaluaron y se decidió usar algunas bibliotecas y utilidades que ayudaran a mejorar la productividad en el proceso de desarrollo. En particular para cubrir requerimientos de portabilidad requeridos en el proyecto y automatizar la construcción e instalación de la solución se utilizaron las Autotools como herramientas de compilación, enlazado y despliegue de la solución. Mientras para el manejo avanzado de memoria se utilizaron `smart pointers` mediante el uso de las bibliotecas **boosts**.

Un factor relevante en el proyecto, por su impácto en la organización del mismo, fue la integración entre los equipos de trabajo que participaron del mismo. Cinco equipos participaron del proyecto, cada uno localizado en diferente zona geográfica y horaria, con grandes diferencias culturales entre ellos, distintos niveles de involucramiento con el lenguaje TTCN-3 y diferentes niveles de experiencia y conocimientos en la construcción de compiladores. Todos estos factores convirtieron en un desafío el trabajo coordinado de estos equipos y la integración de las diferentes partes de la solución.

4.1. Organización del proyecto

Una vez definida la metodológia y diseño general de construcción del compilador, el trabajo fue dividido entre los equipos responsables de cada módulo, estos trabajaron independientemente hasta tener cada módulo operativo.

Para lograr que cada uno trabajase de forma independiente se definió el establecimiento de contratos entre cada módulo. Basados en la estructura del compilador quedarón definidos dos contratos, uno entre el módulo de análisis y el de traducción mediante el AST (Árbol de Análisis

Sintáctico - Abstract Sintax Tree), el otro entre el módulo de traducción y el sistema de tiempo de ejecución mediante el API que este último ofrecería al código traducido, que se llamaron interfaces internas.

Una vez los módulos estuvieron relativamente listos se comenzó el trabajo de integración de los mismos en forma remota, estas tareas de integración tuvieron un avance muy importante durante las jornadas de integración que se realizaron en Beijing, donde gracias a la participación directa de todos los equipos, se pudieron concretar las tareas fundamentales de integración. Las jornadas en Beijing cumplieron un doble rol en este proceso, en primer termino su preparacioón sirvió de motivación para el cierre de cada módulo, pero sin dudas los tres días durante los cuales se discutió y analizó en detalle los mecanismos de intrgración entre los módulos, fueron fundamentales para lograr que el compilador quedase operativo.

Como resultado de este esfuerzo coordinado se construyó el compilador ρTTCN-3, el cual provee una plataforma libre y abierta de compilación y ejecución para TTCN-3, una alta capacidad de adaptación a los dispositivo a testear vía la interfaz de tiempo de ejecución (TRI) definida por el estándar y una sencilla estrategia de integración con CODECS externos mediante la interfaz de control (TCI).

Dos equipos de China y uno de Uruguay participaron en el diseño y desarrollo de los módulos, estos fueron Beijing University of Posts and Telecommunications (BUPT), Inner Mongolia University (IMU) y Universidad de la República - Facultad de Ingeniería (UDELAR/FING). Un equipo de Rusia ISP/RAS trabajó en la validación y aportó en la orientación funcional, un equipo de ARMOR/IRISA/ETSI, localizado en Rennes e integrado por Franceses y Uruguayos contribuyó con la tarea de coordinación, orientación técnica y aseguramiento de la calidad. Una descripción de las áreas de participación de cada universidad y el equipo completo de colaboradores de estas pueden encontrarse en el Anexo 1 15.2.

En la siguiente lista se presenta esquemáticamente la división del proyecto en torno a la construcción de los módulos del compilador y las tareas complementarias como la coordinación y la validación y prueba del compilador:

- ARMOR/IRISA, Coordinación y orientación técnica
- FING/UdelaR, Analizador Sintáctico
- BUPT, Traducción
- IMU , Sistema de tiempo de ejecución
- ISP/RAS, validación y orientación funcional

De acuerdo con esta distribución de tareas, nuestro trabajo principal se concentró en el módulo del analizador sintáctico y léxico, el cual desarrollaremos con más detalle en las siguientes secciones. También colaboramos con BUPT en la etapa de traducción, en primera instancia realizando las transferencia de la solución y *know-how* en esta área, que adquirimos en la primera etapa de desarrollo del compilador en forma solitaria, una vez que BUPT comenzo el desarrollo colaboramos orientando y brindando soporte e ideas para el diseño e implementación de este módulo. Por último, al haber comenzado antes con el desarrollo del compilador ya habíamos definido e implementado la estructura general del mismo, que a la postre fue la utilizada para el compilador ρTTCN-3.

4.2. Diseño general de componentes

Los principales módulos o componentes del compilador son tres, correspondientes a las etapas del mismo definidas por la metodología seleccionada, el Analizador Sintáctico y Léxico, a cargo de la etapa de análisis de la sintaxis cuyo resultado es el árbol de análisis sintáctico (Analytical Syntax Tree - AST), el cual servirá de entrada para la siguiente etapa la Traducción, esta es responsable por la generación del código en el lenguaje destino a partir del AST, en este caso C/C++, código que una vez compilado a código máquina por el compilador de plataforma, en nuestro caso gcc, se enlaza con las bibliotecas correspondientes que al ejecutarse utilizara los servicios del tercer componente de la solución, el sistema de tiempo de ejecución (Run Time System - RTS).

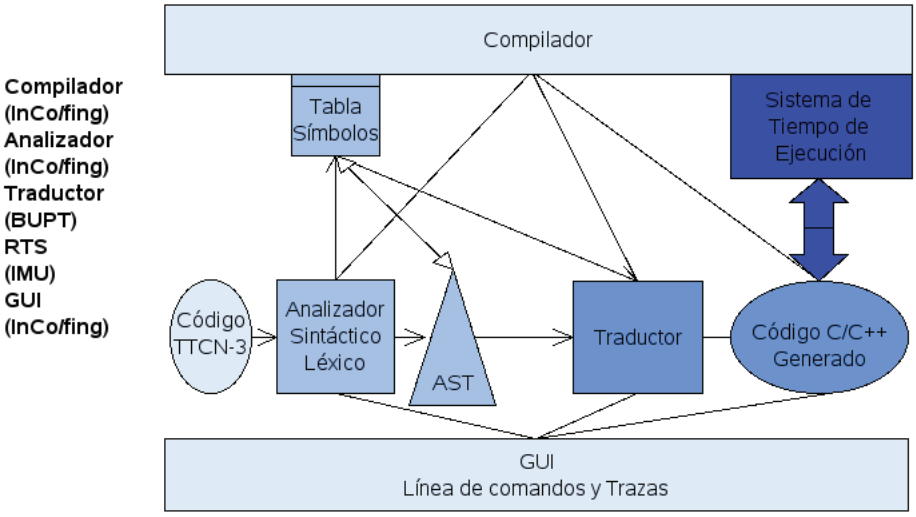

Figura 4.1: Compilador ρTTCN-3 - Diseño general

Además de los módulos correspondientes a cada etapa, hay un módulo principal que comanda el funcionamiento general del compilador y otro especializado en la intereacción con el usuario, a continuación listamos los componentes del compilador y sus responsabilidades:

- Compilador, inicialización y control de la ejecución de las diferentes etapas de la compilación.
- Análizador, análisis sintáctico y léxico, construye la tabla de símbolos y genera como salida el AST.
- Traductor, traducción dirigida por la sintaxis, a partir del AST genera código C/C++.
- Sistema de tiempo de ejecución, bibliotecas para el soporte a la ejecución del código generado por el traductor, en particular soporta la llamada a las interfaces.
- Interacción con el Usuario, maneja los diferentes comandos y los mensajes e información interactiva para quien compila.

Los módulos comúnes al resto de los componentes son el compilador en si mismo y la interacción con el usuario, a continuación realizaremos una breve descripción de estos y en las siguientes

secciones nos detendremos en los tres grandes módulos correspondientes a las etapas del proceso de compilación.

Compilador Es el módulo de más alto nivel de la solución, desde el se maneja el resto de los módulos, es responsable por inicializar algunos parámetros, así como de manejar los archivos de entrada y salida que van a participar del proceso de compilación. Contiene la función principal (main), en esta a partir de la interpretación de los comandos recibidos, serán invocadas las diferentes etapas del compilador hasta la generación del código destino que será en nuestro caso C/C++. Estas etapas son principalmente la fase de análisis sintáctico, que a su vez invoca a demanda al módulo de análisis léxico para obtener los tokens cada vez que los necesita, y posteriormente la fase de traducción donde a partir del árbol sintáctico se realizará la traducción a C/C++.

Interacción con el Usuario Este módulo interpreta los comandos recibidos y cumple con la función auxiliar de generar la información de trazas y errores necesarios para asistir en la etapa de puesta punto, diagnóstico de errores y posteriormente para asistir a los desarrolladores TTCN-3 a compilar sus programas, brindándoles información sobre los errores de codificación detectados por el compilador.

4.3. Diseño del Analizador

Para la etapa de análisis definimos una estructura modular de acuerdo a las buenas prácticas de programación imperativa, ya que si bien evaluamos la posibilidad de trabajar en C++, para utilizar un paradigma orientado a objetos, la dificultad extra que nos presentaban Flex y Bison para esto nos llevó a postergarlo, especialmente en este módulo, para una etapa futura.

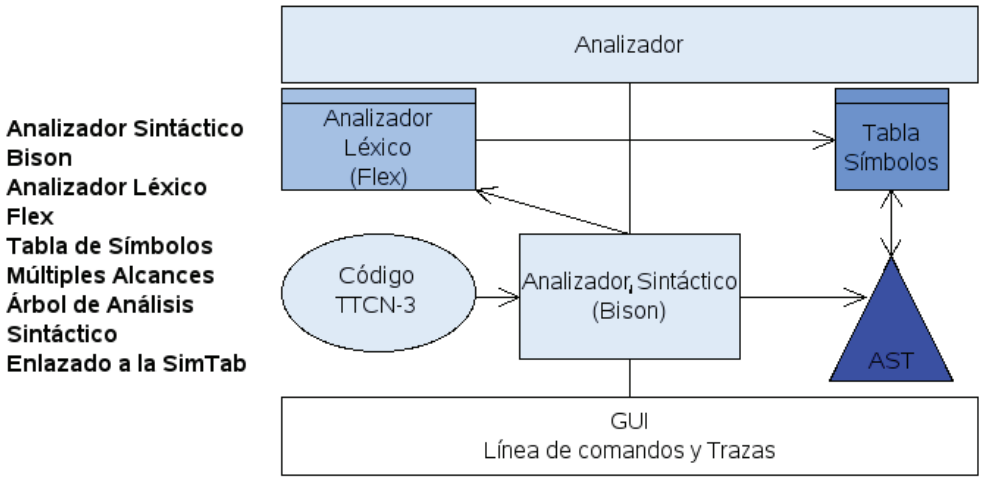

Figura 4.2: Compilador ρTTCN-3 - Diseño Analizador

En el diagrama 4.2 podemos ver los módulos que componen módulo del analizador del compilador:

- Análisis Léxico ***lexico***
- Tabla de Símbolos ***symtab***
- Análisis Sintáctico: ***sintactico***
- Árbol Sintáctico ***ntree***

De acuerdo a las características generales de un Analizador [5], en este diseõ estaría faltando una fase de Análisis Semántico, el mismo quedó fuera del alcance de nuestra implementación, por razones de tiempo y recursos, si bien esto no tuvo ninguna consecuencia operativa su presencia mejoraría las capacidades de la herramienta dotandola, en especial, de un mejor control de tipos. A continuación nos detendremos en cada módulo, explicando las funciones que cumple y como interactua con los otros.

lexico El módulo de análisis léxico está construido usando Flex, a partir de la especificación de las reglas de análisis lexicográfico Flex es capaz de identificar los lexemas válidos del lenguaje, estas reglas se limitan a reconocer patrones del tipo de lenguajes regulares. Se decidió configurar Flex para que el mismo fuera invocado a demanda de Bison de acuerdo a las necesidades de nuevos tokens en el proceso de análisls sintáctico, este es el encare más tradicional y frecuente. Desde el analizador léxico se construye la tabla de símbolos, a medida que se van determinando los nuevos tokens o lexemas se van creando nuevos nodos que se almacenan en la tabla, estos nodos serán luego referenciados desde el árbol sintáctico, y también en los casos que corresponda referenciará al sub-árbol correspondiente a su definición.

symtab Este módulo es el responsable de construir la tabla de símbolos a demanda del analizador léxico. La estructura de la tabla de símbolos tiene la capacidad de manejar distintos alcances para los mismos, para esto se implementó mediante una estructura de pila de arreglos, los arreglos utilizan una tecnica de hashing para optimizar su desempeño y el espacio de almacenamiento utilizado. También ofrece la capacidad de establecer un apuntador al nodo del AST donde se realiza la definición del simbolo.

sintactico De acuerdo a la metodología utilizada el módulo de análisis sintáctico es el que conduce en gran medida, para la etapa de análisis, el proceso de compilación. Basado en la gramática específicada en Bison tratará de reconocer la estructura de las sentencias, invocando al analizador léxico para ir obteniendo los lexemas del código fuente a compilar en la medida que vaya requiriendo estos. Una vez reconocida una sentencia por medio de una regla, mediante el código especificado en las producciones semánticas correspondientes, construye el árbol sintáctico que será la base para la etapa de tradución. Este árbol intentará representar las sentencias del lenguaje reconocidas en el código fuente, de forma de facilitar el trabajo de la fase de producción. El módulo de análisis sintáctico fue construido utilizando Bison, esto se hace especificando las reglas de reconocimiento de la gramática, de acuerdo las posibilidades ofrecidas por Bison y las producciones semánticas asociadas a las mismas. Para la escritura de estas reglas, se desarrollaron y utilizaron algunas herramientas de generación automática a partir del BNF del lenguaje y también algunos criterios o reglas para la generación del AST. Gracias al uso de estas herramientas se redujo significativamente el tiempo necesario para la especificación de dichas reglas, este tema será desarrollado con mayor detalle en la sección 5.1.

ntree Este módulo es el responsable de la gestión del árbol de análisis sintáctico, el mismo permite construcción y recorrida del mismo, fue diseñado para contener toda la informacón necesaria

para la traducción así como para los chequeos semánticos correspondientes. Su estructura se optimizó para facilitar y mejorar la traducción, para esto se incluyeron referencias a los nodos de la tabla de símbolos, cuando esto era necesario, también se agregarón referencias a sub-árboles, cuando se requería tener una definición relacionada a un nodo disponible rápidamente. A efectos de la verificación se incluyó una rutina de recorrida e impresión del árbol sintáctico, dicha rutina se constituyó en el punto de partida del algoritmo de recorrida para la estapa de traducción.

4.4. Diseño del Traductor

El traductor es el responsable de recorrer el AST y generar el código destino, para esto tiene en cuenta las interfaces internas definidas con el RTS, utilizando de esta forma desde el código traducido a C/C++ los servicios brindados por este.

La fase de traducción se apoyó fuertemente en el trabajo ya adelantado por nosotros para la impresión del AST, a efectos de nuestras pruebas, por lo tanto se reutilizó la lógica general propuesta originalmente, la misma consistía en una recorrida recursiva del AST.

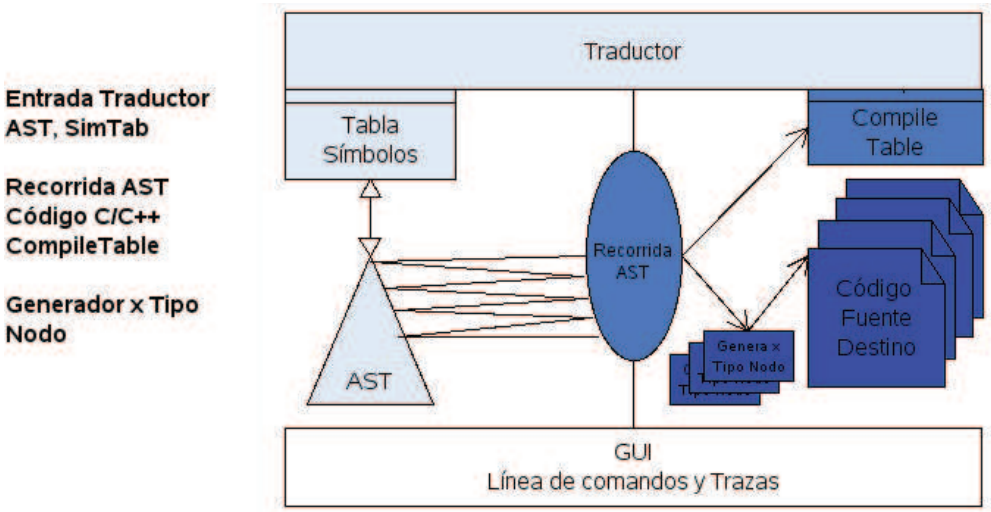

Figura 4.3: Compilador ρTTCN-3 - Diseño Traductor

Este proceso se realiza en dos etapas, una para la generación de una estructura de lista que contenga todos los elementos de la parte de comportamiento del código TTCN-3 (BehavorObjectList) y una segunda que realiza la generación de todas las declaraciones y definiciones.

En la primera etapa se recorre el árbol en profundidad, primero los hijos y luego los hermanos, agregando a la BehavorObjectList cada Caso de Prueba con información del componente y las interfases que utiliza. A partir de dicha lista se generará la clase CompileTable, la cual contiene las llamadas a los Casos de Pruebas (TestCases), que luego nos permitirá conocer que casos de prueba están disponibles en un componente.

Mientras que la segunda etapa se basa en un procedimiento general de recorrer el árbol, para luego con rutinas especializadas de impresión de los diferentes tipos de nodos realizar recorridas parciales de sub-árboles de acuerdo a la información necesaria para cada tipo de nodo, también en los casos que esto es necesario se recurre a sub-árboles con definiciones o la información disponible

en la lista de símbolos.

4.5. Diseño del RTS

El RTS tiene como responsabilidad proveer cualquier elemento de infraestructura o interfaz de acceso a la misma, que sea necesario para que, el programa producido a partir del código TTCN-3, se pueda ejecutar en una plataforma de base. En el caso de TTCN-3 esto incluye brindar soporte para las diferentes interfaces definidas en el estándar. En particular se tuvo en cuenta para el diseño de las clases del RTS la estructura del lenguaje definida por el estándar, las interfaces que esté debía soportar para las implementaciones y otros elementos como la descomposición en componentes que realiza el estándar. En la figura 4.4 presentamos la arquitectura del compilador ρTTCN-3 donde se encuentran detallados los componentes del RTS dentro del TE, entorno al C-ATS, en particular se nota la relación para la implementación de las interfaces.

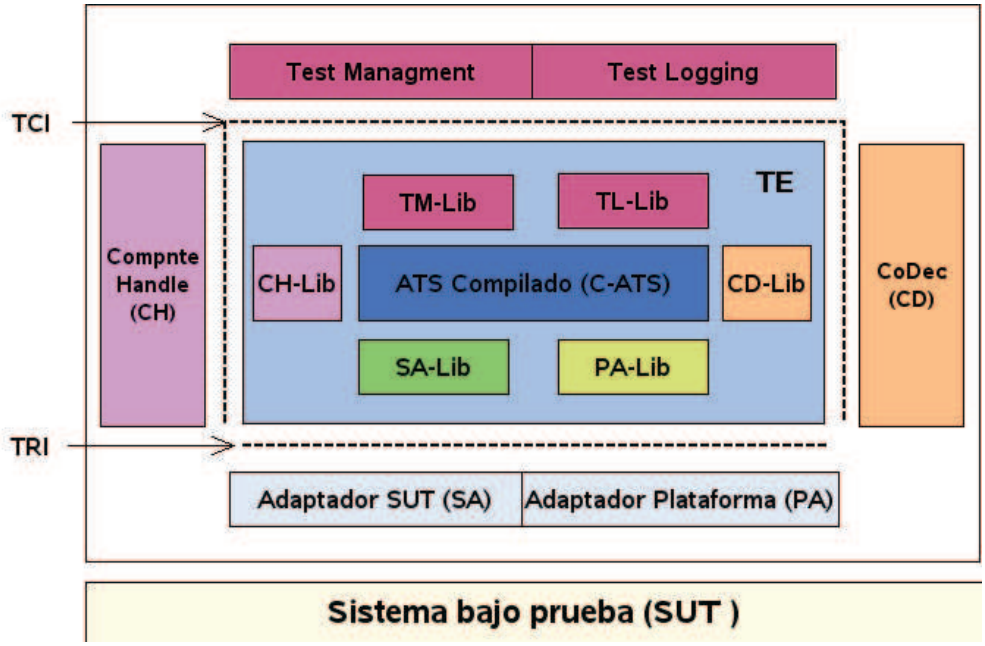

Figura 4.4: Compilador ρTTCN-3 - Diseño RTS

En el caso de RTS, al ser un componente completamente segregado y haber contado con un equipo independiente y numeroso de IMU, se pudo aplicar de forma bastante consistente el paradigma de objetos para su diseño, de esta forma se modelaron las distintas clases siguiendo las definiciones dadas por el estándar. Las figuras 4.5 y 4.6 muestran los diagramas de clase del T3RTS donde pueden identificarse las relacione de generalización, composición, agregación y dependencia respectivamente.

4.6. Validación de la gramática aceptada por el compilador

El equipo ruso de ISP/RAS aportó metodología y herramientas para la validación automática de la gramática aceptada por el compilador. La técnica utilizada se basa en el trabajo "Automated Generation of Positive and Negative Tests for Parsers" [42], el mismo propone la generación

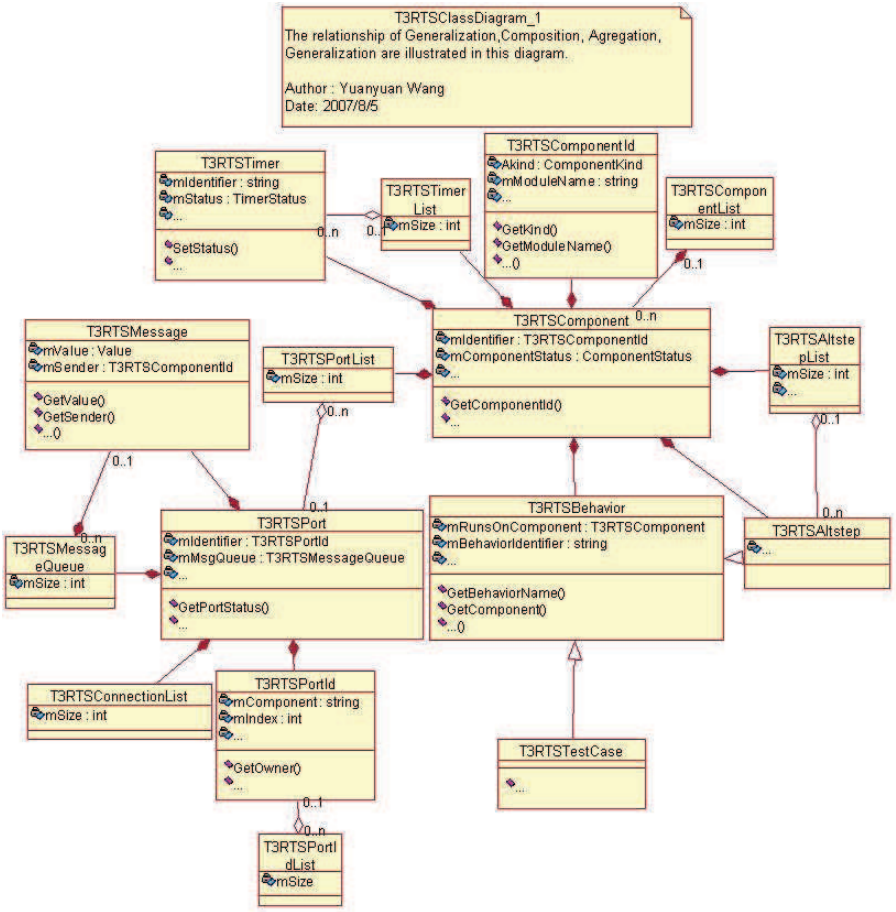

Figura 4.5: Compilador ρTTCN-3 - Diseño RTS Relaciones de Generalización, Composición y Agregación

automática de casos de prueba positivos y negativos, a partir de la especificación de la gramática del parser. En este trabajo se propone y muestran resultados sobre un nuevo criterio para la definición del conjunto de cobertura de las pruebas, un aspecto destacado de este es la utilización de casos negativos, cosa no habitual en este tipo de técnicas. Se realiza un análisis sobre la efectividad del mismo mostrando de esta forma la validez de los resultados, se presentan finalmente algunos resultados prácticos.

La validación de la gramática para el reconocimiento del lenguaje es una parte esencial del testing de un compilador, la correctitud del resto del compilador se sustenta en que la etapa de análisis este correctamente especificada. Si bien quedaron por cubrir algunos aspectos de las pruebas y validaciones, necesarias en el proceso de testing del compilador, la tarea acometida en conjunto con ISP/RAS resuelve una parte fundamental de este, que demandaría una gran cantidad de trabajo si se realizase de otro modo. La experiencia de ISP/RAS en esta área nos brindó, además del trabajo concreto que ellos realizaron generando los casos de prueba, la tranquilidad de que nuestro trabajo en el área del Analizador estaba correctamente orientado. El proceso de pruebas realizadas nos permitió detectar y corregir un conjunto de errores en la especificación de la gramática, también algunas de las pruebas que fallaron fueron descartadas luego de su análisis ya que no respondían a errores reales sino a características de la técnica empleada.

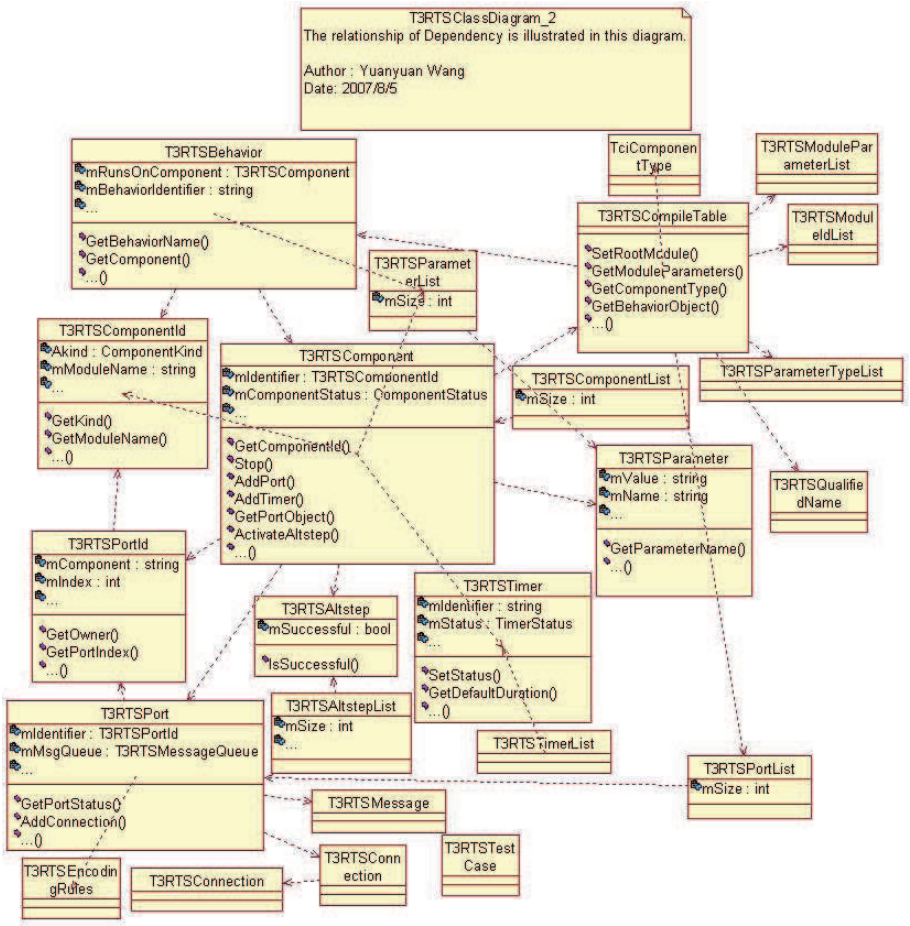

Figura 4.6: Compilador ρTTCN-3 - Diseño RTS Relaciones de dependencia

Capítulo 5

Decisiones de Implementación

En esta sección abordamos algunas decisiones relacionadas con la implementación, especialmente con las herramientas utilizadas en el desarrollo de las diferentes etapas del compilador, también brindamos una especificación más formal del alcance de la implementación basados en el conjunto de palabras reservadas o tokens de TTCN-3 y en la cobertura de la gramática.

5.1. Herramientas de Desarrollo

Como establecimos en la sección 3.3, si bien la investigación de nuevas tecnologías disponibles para la construcción de compiladores, no nos llevó a seleccionar una herramienta específica complementaria a Bison/Flex para mejorar la automatización del proceso, si nos dejó algunas ideas que pudimos aplicar prácticamente durante el proceso de construcción del compilador.

Lo más interesante, por el volumen de trabajo ahorrado, fué la posibilidad de inferir o derivar automáticamente desde la especificación de la gramática del lenguaje en BNF las especificaciones de las reglas y sus acciones para Bison y Flex.

Con este objetivo desarrollamos un programa que a partir del BNF de la gramática de TTCN-3 como entrada genera las reglas de análisis para Bison, el mismo cuenta con varias etapas que permiten inferir los diferentes componentes de la especificación de forma incremental. En general cada etapa usa en parte el resultado de las anteriores y la división realizada permite intercalar cierto trabajo manual para complementar lo generado automáticamente antes de pasar a la siguiente etapa.

El mismo criterio, aunque en un problema más simple, fue aplicado para las reglas de análisis léxico, por lo cual se contruyó un módulo que a partir de las expresiones regulares que especificaban los lexemas genera las especificaciones para flex. El Diagrama 5.1 muestra las diferentes etapas y las tareas que se realizan en estas.

En una etapa posterior y basados en la experiencia de las primeras etapas del desarrollo percibimos un alto nivel de regularidad en las producciones o acciones semánticas que agregábamos relacionadas a las reglas, esto nos llevó a automatizar también en gran medida las producciones semánticas, quedando solo la revisión del resultado y la corrección de algunos detalles y casos no cubiertos para especificar manualmente.

Las Figuras 5.2, 5.3 y 5.4 muestran las diferentes etapas en que se van generando las especificaciones.

Etapa 1, Generación de las declaraciones para Bison y Flex a partir del BNF, también genera las reglas y acciones para Flex

Etapa 2, Generación de las reglas y acciones básicas para Bison a partir del BNF y las declaraciones

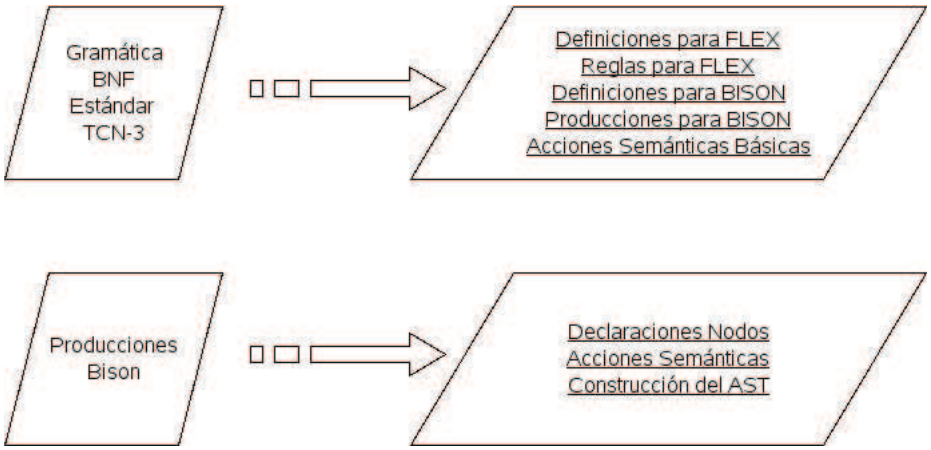

Figura 5.1: Desarrollo - Etapas del Asistente

Etapa 3, Generación de acciones para la construcción del AST a partir de los cabezales de declaraciones y la gramática básica ya generada.

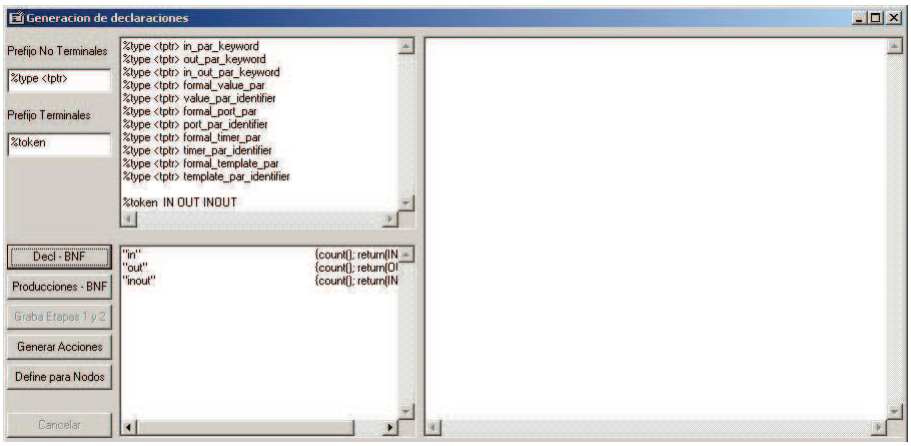

Figura 5.2: Asistente Etapa 1 - Genera declaraciones desde BNF

El proceso de desarrollo de esta herramienta fue guiado por un sentido muy pragmático y concreto para nuestra implementación, si bien esta característica le quita generalidad nos permitió encontrar el mejor balance entre el esfuerzo de desarrollar la herramienta y el volumen de trabajo que esta lograba automatizar.

En nuestra opinión la etapa más interesante fue la tercera, las otras dos están más relacionadas a la conversión de las especificaciones en BNF a Bison, en cambio la última implicó establecer reglas para la construcción del AST a partir de la estructura de cada producción y los tipos de los símbolos terminales y no terminales, en este sentido podemos ver los siguientes casos donde se genera código de acuerdo a estos criterios.

- No terminal Izquierdo
  - Si tiene hijos y no es una lista Se crea el nodo y se agregan los hijos GenAddHijos(sLin, sLTknType)

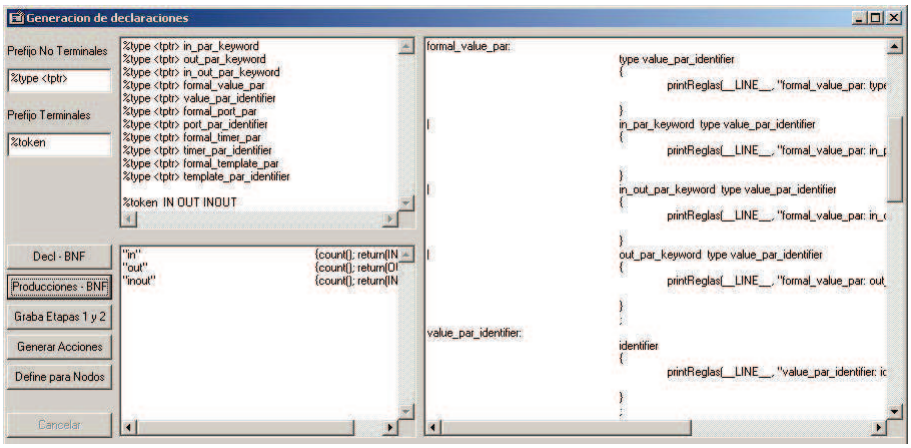

Figura 5.3: Asistente Etapa 2 - Genera acciones básicas desde BNF

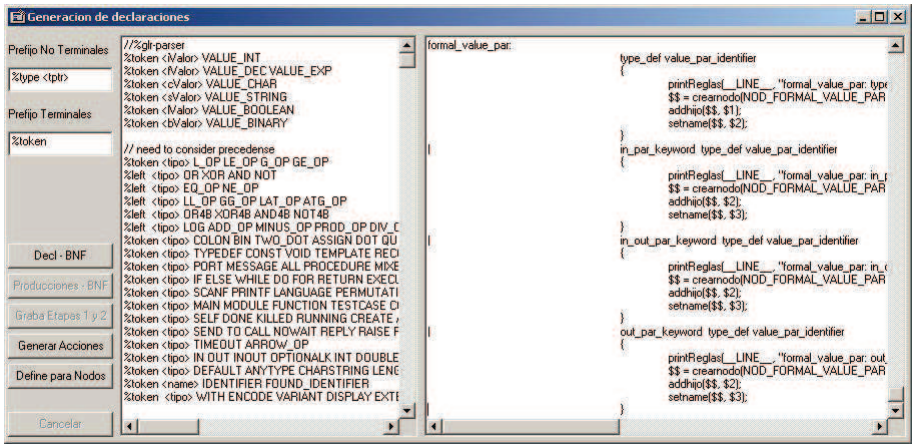

Figura 5.4: Asistente Etapa 3 - Genera acciones para construcción del AST

- Si tiene hijos y es una lista Se agrega el siguiente hermano
- Si es un tipo se asigna al padre
- Si es un name se setea el nombre al padre
- Si es un name compuesto se setean los nombres al padre
- Si es un valor (entero, cadena, lógico, flotante o binario) se asigna al padre

- No terminal Derecho, al agregar hijos
  - si también tendrá hijos agrega hijos
  - Si es un name se setea el nombre al padre
  - Si es un name compuesto se setean los nombres al padre
  - si es tipo y se conoce el mismo se setea el tipo sino no se setea
  - Si es un valor (entero, cadena, lógico, flotante o binario) se setea el valor con las funciones correspondientes setinteger, setstring, setboolean, setfloat, setstring

A continuación mostramos un ejemplo de la generación de los diferentes elementos mencionados para una pequeña sección de la gramática expresada en el BNF del estándar.

5.1.1. Especificación del BNF en el estándar

```
501. InParKeyword ::= "in"
502. OutParKeyword ::= "out"
503. InOutParKeyword ::= "inout"
504. FormalValuePar ::= [(InParKeyword | InOutParKeyword |
OutParKeyword)] Type ValueParIdentifier
505. ValueParIdentifier ::= Identifier
506. FormalPortPar ::= [InOutParKeyword] PortTypeIdentifier
PortParIdentifier
507. PortParIdentifier ::= Identifier
508. FormalTimerPar ::= [InOutParKeyword] TimerKeyword
TimerParIdentifier
509. TimerParIdentifier ::= Identifier
510. FormalTemplatePar ::= [(InParKeyword | OutParKeyword |
InOutParKeyword )] TemplateKeyword Type TemplateParIdentifier
511. TemplateParIdentifier ::= Identifier
```

5.1.2. Reglas y acciones para Flex

```
"in" {count(); return(IN);}
"out" {count(); return(OUT);}
"inout" {count(); return(INOUT);}
```

5.1.3. Declaraciones para Bison y Flex

```
%type <tptr> in_par_keyword
%type <tptr> out_par_keyword
%type <tptr> in_out_par_keyword
%type <tptr> formal_value_par
%type <tptr> value_par_identifier
%type <tptr> formal_port_par
%type <tptr> port_par_identifier
%type <tptr> formal_timer_par
%type <tptr> timer_par_identifier
%type <tptr> formal_template_par
%type <tptr> template_par_identifier

%token  IN OUT INOUT
```

5.1.4. Reglas y acciones básicas para Bison

```
in_par_keyword:
"in"
{
```

```
printReglas(__LINE__, "in_par_keyword: "in"");

}
;
out_par_keyword:
"out"
{
printReglas(__LINE__, "out_par_keyword: "out"");

}
;
in_out_par_keyword:
"inout"
{
printReglas(__LINE__, "in_out_par_keyword:
"inout"");

}
;
formal_value_par:
type value_par_identifier
{
printReglas(__LINE__, "formal_value_par: type
value_par_identifier");

}
| in_par_keyword  type value_par_identifier
{
printReglas(__LINE__, "formal_value_par:
in_par_keyword  type value_par_identifier");

}
| in_out_par_keyword  type value_par_identifier
{
printReglas(__LINE__, "formal_value_par:
in_out_par_keyword  type value_par_identifier");

}
| out_par_keyword  type value_par_identifier
{
printReglas(__LINE__, "formal_value_par:
out_par_keyword  type value_par_identifier");

}
;
value_par_identifier:
identifier
{
```

```
printReglas(__LINE__, "value_par_identifier:
identifier");

}
;
```

5.1.5. Acciones para construcción del AST

```
in_par_keyword:
"in"
{
printReglas(__LINE__, "in_par_keyword: "in"");
$$ = $1;
}
;
out_par_keyword:
"out"
{
printReglas(__LINE__, "out_par_keyword: "out"");
$$ = $1;
}
;
in_out_par_keyword:
"inout"
{
printReglas(__LINE__, "in_out_par_keyword:
"inout"");
$$ = $1;
}
;
formal_value_par:
type_def value_par_identifier
{
printReglas(__LINE__, "formal_value_par: type
value_par_identifier");
$$ = crearnodo(NOD_FORMAL_VALUE_PAR);
addhijo($$, $1);
setname($$, $2);
}
| in_par_keyword  type_def value_par_identifier
{
printReglas(__LINE__, "formal_value_par:
in_par_keyword  type_def value_par_identifier");
$$ = crearnodo(NOD_FORMAL_VALUE_PAR);
addhijo($$, $2);
setname($$, $3);
}
| in_out_par_keyword  type_def value_par_identifier
```

```
{
printReglas(__LINE__, "formal_value_par:
in_out_par_keyword  type_def value_par_identifier");
$$ = crearnodo(NOD_FORMAL_VALUE_PAR);
addhijo($$, $2);
setname($$, $3);
}
| out_par_keyword  type_def value_par_identifier
{
printReglas(__LINE__, "formal_value_par:
out_par_keyword  type_def value_par_identifier");
$$ = crearnodo(NOD_FORMAL_VALUE_PAR);
addhijo($$, $2);
setname($$, $3);
}
;
value_par_identifier:
identifier
{
printReglas(__LINE__, "value_par_identifier:
identifier");
strcpy($$, $1);
}
;
```

Por otro lado se automatizaron algunos aspectos relacionados con el proceso de desarrollo en si mismo que mostraron ser muy demandantes de tiempo, guiados por la necesidad nos enfocamos en algunos procesos repetitivos del ciclo de desarrollo, como la revisón de la gramática ya escrita a efectos de corregir errores, realizar modificaciones y liberar nuevas versiones. Dado a que este proceso fue incremental y durante el mismo debímos interactuar, realizando varias iteraciones, con los equipos que trabajaban en la traducción y sistema de tiempo de ejecución, nos vimos en la necesidad de agregar paulatinamente nuevas acciones semánticas al parser y en algunos casos reescribir parte de las producciones semánticas para enriquecer el AST, agregando algunas características requeridas por estos equipos. Para cumplir con estos requerimientos debimos escribir algunos programas, que llamamos filtros, con la capacidad de complementar las producciones semánticas y también de trabajar solo sobre una parte de las definiciones.

Como resultado de este proceso nos quedamos con un conjunto de utilitarios que automatizando las tareas descritas redujeron el trabajo manual en varios ordenes, convirtiendose nuestra mayor tarea en complementar algunas especificaciones, validar el resultado obtenido y poner a punto las especificaciones del parser. Otro factor importante, que justificó la inversión de tiempo en la automatización de estas tareas, fue la disminución en la taza de errores al generar o regenerar las especificaciones de esta forma, brindando así a los demás equipos un importante nivel de estabilidad en la solución manejada.

Dentro de las utilidades que construimos, para la validación del AST generado, se encontraban diferentes rutinas de impresión para el árbol sintáctico, las cuales fueron utilizadas luego como base para la construcción del algoritmo de recorrida del AST del traductor.

5.2. Alcance de la implementación

Como explicamos en 2, quedaron establecidos dos niveles en el alcance de la solución, que ahora formalizamos como resultado de la implementación. El alcance inicial dado por nuestro análisis de uso del lenguaje, basado en las estadísticas de uso de las keywords del mismo por parte de ARMOR/IRISA, esto se ve reflejado en la Tabla 1 5.1 y un segundo nivel dado por la definición de alcance de Go4IT, en base a las necesidades de este proyecto, llamada A0 y expresada en función de la cobertura de la gramática a través de su especificación en BNF, esta por ser demasiado extensa se deja disponible en el Anexo 2.

Keyword		**Keyword**		**Keyword**		**Keyword**	
action	N	encode	Y	language	N	read	Y
activate	N	enumerated	Y	length	Y	receive	Y
address	Y	error	N	log	Y	record	Y
alive	N	except	N	map	Y	rem	N
all	Y	exception	N	match	Y	repeat	N
alt	Y	execute	Y	message	Y	reply	Y
altstep	Y	extends	N	mixed	Y	return	Y
and	Y	Extension	N	mod	Y	running	N
and4b	Y	external	N	modifies	Y	runs	Y
any	Y	fail	Y	module	Y	select	Y
anytype	Y	false	Y	modulepar	Y	self	Y
bitstring	Y	float	Y	mtc	Y	send	Y
boolean	Y	for	N	noblock	Y	sender	Y
case	Y	from	Y	none	Y	set	Y
call	Y	function	Y	not	Y	setverdict	Y
catch	N	getverdict	Y	not4b	Y	signature	Y
char	Y	getcall	Y	nowait	Y	start	Y
charstring	Y	getreply	Y	null	Y	stop	Y
check	Y	goto	Y	octetstring	Y	subset	N
clear	Y	group	Y	of	Y	superset	N
complement	N	hexstring	Y	omit	Y	system	Y
component	Y	if	Y	on	Y	select	Y
connect	Y	ifpresent	N	optional	Y	self	Y
const	Y	import	N	or	Y	send	Y
control	Y	in	Y	or4b	Y	sender	Y
create	Y	inconc	Y	out	Y	set	Y
Deactivate	N	infinity	Y	override	Y	setverdict	Y
Default	Y	inout	Y	param	Y	signature	Y
disconnect	Y	integer	Y	pass	Y	start	Y
display	Y	interleave	N	pattern	Y	stop	Y
do	Y	kill	N	port	Y	subset	N
done	Y	killed	N	procedure	Y	superset	N
else	Y	label	N	raise	N	system	Y

Tabla 5.1: Tabla de cobertura

De acuerdo a lo establecido en 2.2 el segundo un alcance propuesto, más amplio y adecuado a las necesidades de Go4IT fue tomado como el definitivo y solo se consideraron en algunos casos, significativos por el volumen de trabajo ahorrado, las limitantes definidas en el primero, por ejemplo respecto a la implementacińn de los tipos de datos.

Existieron otros limites a la implementación relacionados con el alcance de las tareas del Traductor o el RTS, que tienen que ver con la implementación de ciertos aspectos del funcionamiento del lenguaje que quedaron pendientes por su complejidad y no ser requeridos para Go4IT. Para conocer más en detalle la implementación realizada, los problemas que enfrentó y las soluciones encontradas referirse al Anexo 3.

Capítulo 6

Evaluación del trabajo en el compilador ρTTCN-3

En este capítulo revizaremos las leccciones aprendidas durante la construcción del compilador, presentaremos algunos temas pendientes en el mismo que pueden ser de interés para su uso en otros contextos y finalmente realizaremos una evaluación del proyecto de construcción de ρTTCN-3.

6.1. Lecciones aprendidas

Dentro de las lecciones aprendidas en este proceso podemos destacar por un lado el papel de las herramientas de asistencia, complementarias a Bison/Flex en la construcción del compilador, incluido el desarrollo y características de las mismas, por otro el desafío enfrentado en el proyecto debido al trabajo con varios equipos, altamente heterogéneos y distribuidos geográficamente.

6.1.1. Herramientas de desarrollo

Por su impacto en la factibilidad del proyecto, el desarrollo y la utilización de las herramientas de asistencia, para la generación de las especificaciones para Bison/Flex a partir del BNF de la gramática, resultaron uno de los factores más relevantes de esta experiencia.

En primer termino redujeron en más de un orden el trabajo repetitivo de la codificación de las reglas y acciones semánticas correspondientes, esto resulta muy adecuado especialmente en la construcción de una base experimental, donde algunos detalles no son tan relevantes como en una solución para la industria, y por lo tanto es más sencillo generalizar comportamientos, de todas formas los mismos conceptos podrían utilizarse para generar una versión inicial que luego se refinaría manualmente.

Para ofrecer una referencia que permita cuantificar el esfuerzo ahorrado mencionamos a continuación el tamaño de las especificaciones generadas. La especificación para Bison tiene 8500 líneas (archivo sintactico.y), de las cuales menos del 10 % son código de soporte escrito en forma manual, mientras el resto fué generado por las herramientas mencionadas y está compuesto por declaraciones, reglas de reconocimiento de la gramática y producciones semánticas. La especificación para Flex está compuesta por 350 líneas (archivo lexico.l), de las cuales menos del 20 % fueron escritas manualmente. Como contrapartida se generaron un conjunto de rutinas que en su totalidad no superan las 800 líneas de código y donde apróximadamente el 50 % son de soporte para manipular archivos y parsear las cadenas de caracteres. Además es importante destacar que en el caso de la especificación de Bison ésta tuvo varias pasadas, donde se corrigieron y agregaron especificaciones que implicaron cambios menores en la herramienta pero impactaron en toda la especificación.

En segundo termino la utilización de las herramientas colabora en la generación de un árbol sintáctico más homogeneo y estructurado, debido a la necesidad de factorizar los elementos comunes y convertirlos en reglas para la generación de la espcificación de Bison/Flex y las producciones semánticas relacionadas, tratando de forma consistente todos los casos de producciones similares o con el mismo formato. Garantizar esta consistencia trabajando de forma manual requeriría mayor trabajo de coordinación, sobre todo, trabajando en equipos grandes y distribuidos.

Por último, debido a la dinámica de trabajo que exigió la realización de modificaciones generales en varias oportunidades, se logró también la capacidad de automatizar el reproceso de las reglas, permitiendo corregir de forma genérica elementos del árbol sintáctico o introducir modificaciones a las producciones semánticas con un mínimo de retrabajo y de introducción de nuevos errores.

Otra enseñanza que podemos extraer, relacioanda con las herramientas generadas en si mismas, es que estas fueron acompañando el proceso de desarrollo de una forma extremadamente *ad hoc*, especialmente al momento de decidir donde detener el esfuerzo de solucionar algunos casos automáticamente y pasar a su resolución manual, esto nos reafirmó que hubiese resultado más difícil reusar herramientas de terceros donde no hubiese sido tan simple realizar esta opción.

En este sentido vemos como una alternativa más práctica, a las herramientas que evaluamos inicialmente para asistirnos en el desarrollo del compilador, la existencia de bibliotecas que den soporte a algunas tareas que se cubrieron con las herramientas *ad hoc*, un ejemplo claro puede ser la búsqueda y transformación de patrones en cadenas de caracteres, este trabajo muy costoso de realizar en herramientas de plataforma puede abstraerse fácilmente en bibliotecas y mejorar la productividad de estas tareas. Mucho de este trabajo podría realizarse con las mismas herraientas Flex y Bison u otra herramienta que trate con expresiones regulares, en lugar de la técnica utilizada de programaciń tradicional, para determinar cual es la mejor alternativa deberían evaluarse los costos relativos y la flexibilidad resultante.

6.1.2. Organización del proyecto

Otro aspecto no tan técnico pero de mucho impacto en el proyecto fue la experiencia de trabajo con varios equipos distribuidos geográficamente y con contextos culturales y académicos muy diferentes, este se constituyó en un desafío interesante que nos dejó varias enseñanzas. En un entorno con estas características se vuelven más difíciles y se necesitan mucho más, las tareas de coordinación y comunicación, diferentes aspectos que pueden darse por sobreentendidos, en equipos más homogéneos, deben establecerse de forma explícita en esta situación, y requerirse compromisos claros y formales de las diferentes partes para abordarlos. En entornos distribuidos el conocimiento personal mejora sustancialmente la comunicación, así como los niveles de colaboración y compromiso con los objetivos, de la misma forma las tareas de integración se multiplican al conocerse claramente y explicitarse lo más directamente posible las diferentes responsabilidades. Esto se reflejó claramente luego de las instancias presenciales, como la de Beijing descrita en 4.1, donde además de establecerse un conocimiento personal entre los integrantes de los diferentes equipos se alinearon las expectativas y necesidades del proyecto, se notaron diferencias importantes en el desempeño del trabajo de los equipos y en la capacidad de trabajar de forma integrada, logrando más y mejores resultados concretos.

Otro aspecto que requiere más atención, en este contexto, es la definición y publicación de la documentación técnica, esta se convierte en una herramienta imprescindible para evitar situaciones de dependencia no deseadas, en la medida de lo posible deben establecerse de acuerdo a estándares internacionales conocidos y aceptados por todas las partes. La participación de recursos poco experimentados también tiene importantes consecuencias a tomar en cuenta, en muchos casos la curva de aprendizaje puede ser mayor a lo esperado, requiriendo un alto grado de supervisión y apoyo

para lograr niveles de productividad aceptables, como contrapartida este tipo de recursos tiene una alta permeabilidad para adoptar metodologías y técnicas de trabajo nuevas y seguir las pautas de trabajo propuestas, estos elementos considerados en forma conjunta mostraron claramente la necesidad de realizar mayores tareas de orientación y transmisión de conocimientos de las previstas originalmente.

6.2. Trabajos pendientes

Visto desde una perspectiva general podríamos plantear muchas mejoras para la implementación que quedó disponible del compilador, pero dado al cometido específico con el cual se desarrolla el mismo nos concentraremos en las mejoras que de acuerdo a este serían necesarias, para esto las dividiremos en técnicas y funcionales.

6.2.1. Mejoras Técnicas

De acuerdo a las definiciones originales realizadas en la sección 4, queda pendiente migrar los módulos correspondientes a las etapas de Análisis Sintáctico y Lexicográfico de C a C++, esto implicará un rediseño con orientación a objetos, para esto se debe realizar un estudio y análisis más profundo de los requerimientos de las herramientas Flex/Bison a este respecto. También en este sentido debería trabajarse con el módulo que implementa la etapa de traducción.

En el área de la validación y pruebas del compilador, se deberá extender la cobertura de las pruebas a todas las áreas del lenguaje, hoy las pruebas más formales se restringen al analizador, el resto de las etapas se ha probado para algunos ejemplos de interés del proyecto, pero de forma no exhaustiva y sin una orientación metodológica bien definida. Dentro de las pruebas del analizador y de acuerdo a la metodología y herramientas adoptadas aún está pendiente cubrir la mayor parte de las pruebas de los casos negativos.

6.2.2. Funcionalidades Pendientes

Desde la óptica del compilador, la etapa de Análisis Semántico quedó pendiente en esta implementación, la misma deberá realizarse si se quiere contar los controles correspondientes, dotando al compilador de capacidades como la detección de errores de tipos en etapas tempranas.

Desde una perspectiva del lenguaje faltan desarrollar varias capacidades complementarias del compilador como Logging, Control Managment y otras que se encuentran expuestas en la TCI. También algunas reglas de la gramática más complejas, como las declaraciones anidadas y características como la importación de módulos deberán evaluarse para su incorporación.

Opcionalmente se debería considerar si se dará cobertura en el compilador a algunas características avanzadas del lenguaje, que fueron dejadas de lado por consideraciones relacionadas a la metodología utilizada generalmente en ARMOR/IRISA 2, como la ejecución distribuida y el soporte a algunos tipos de datos avanzados.

6.3. Evaluación del proyecto ρTTCN-3

El proyecto para la construcción del compilador ρTTCN-3 tuvo como principal objetivo crear una plataforma TTCN-3 para la investigación y prueba del lenguaje, se cumplió con este objetivo en la medida que se liberó, dentro de los plazos establecidos en el proyecto, una versión operativa para el alcance de la gramática definida como A0, la cual está especificada en el Anexo 2.

Esta versión fue utilizada en primera instancia para realizar experimentación con la Solución Dual, la cual brinda portabilidad a las implementaciones de TTCN-3 entre diferentes plataformas de base.

Respecto a la evaluación del compilador como producto, podemos hacer referencia a las pruebas realizadas a la gramática aceptada o rechazada por el mismo en base a las herramientas de ISP/RAS presentadas en 4.6, las cuales generan de forma automática casos de prueba positivos y negativos en base al BNF del lenguaje. El resultado de estas pruebas que evaluaban el compilador desde la perspectiva de la gramática fue muy positivo, permitiendó poner a punto el analizador sintáctico, corrigiendo los errores detectados con esta técnica y llegando a pasar todas las pruebas generadas que se consideraron correctas.

Por otro lado para una prueba funcional completa quedó pendiente la formalización y ejecución de un proceso de testing que abarcara las demás etapas del compilador. Las pruebas realizadas a las demas etapas se limitaron a las necesarias para la realización de la prueba de concpetos planteada para el proyecto, en especial la ejecución de algunos casos de pruebas de conformidad con IPv6 de la suite de Go4IT.

Otros tipos de pruebas mas específicas como usabilidad o desempeño no llegaron a plantearse en el contexto del proyecto por razones de tiempo y recursos, aunque estos aspectos si fueron considerados a la hora del diseño y construcción del compilador y por lo tanto deberían formar parte del plan de pruebas del mismo.

6.3.1. Repercusiones del trabajo

Además del uso experimental que dimos al compilador ρTTCN-3, este también despertó el interés de la comunidad ρTTCN-3 que se ha comunicado para consultar por su estado y sus capacidades, con el objetivo de utilizarlo en otros proyectos. En el período inmediato posterior al proyecto recibimos numerosas consultas de la comunidad TTCN-3, sobre la posibilidad de usar ρTTCN-3 para diferentes aplicaciones y proyectos.

Dado este interés propusimos al equipo de Go4IT del proyecto para la construcción del compilador ρTTCN-3, presentar un trabajo sobre la construcción del compilador abierto y sus resultados en la Conferencia de Usuarios TTCN-3 de 2010 en Beijing, China [19]. La propuesta fue aceptada y en forma colectiva presentamos el trabajo: **"An open compiler for TTCN-3: picoTTCN-3"** [33]. Dado que la conferencia se realizaba en Beijing-China la presentación quedó a cargo de una integrante china del equipo de Go4IT, Xiaohong Huang. La presentación recibió diferentes expresiones de interés de la comunidad asiática de TTCN-3, la que mayormente se encontraba en la conferencia. Destacando en este sentido Huawei, RFI China, ZHONGCHUANG TELECOM TEST, Tsinghua University, entre otras.

En el caso particular de la empresa Huawei, la cual está trabajando en el desarrollo de su propia plataforma TTCN-3, se interesaron por el estado del proyecto y como resultado de la información obtenida están evaluando la posibilidad de re-utilizar, en su proyecto interno, el trabajo realizado en ρTTCN-3 [25].

Parte II

Una solución multiplataforma para TTCN-3

Capítulo 7

Motivación y Objetivos de la dualización

Una de las fortalezas de TTCN-3 es su capacidad de definir Pruebas Abstractas (Abstract Test Suites - ATS), para esto el lenguaje ofrece un mecanismo para eliminar los detalles propios de la plataforma de base en la definición y el diseño de las pruebas, como desarrollamos en el capítulo de introducción al lenguaje TTCN-3 1.1.

Entre otras ventajas, esta característica facilita al diseñador de las pruebas concentrarse en la diseño de éstas, dejando de lado los detalles de la implementación en el lenguaje de plataforma, mejorando al mismo tiempo la generalidad y portabilidad de los casos de prueba definidos. Esto permitiría, al menos teóricamente, que la defínción de un caso de prueba en TTCN-3 fuese utilizada para generar implementaciones con diferentes herramientas en ambas plataformas, sin necesidad de realizar cambios en el código fuente TTCN-3.

Como contrapartida, al momento de construir los ejecutables de los casos de prueba (Test Executable - TE), como se explicó en 1.1, la información dependiente de la plataforma debe ser provista para completar la implementación. Esto se realizará de alguna forma dependiente de la plataforma, durante la implementación de los Casos de Prueba deberán proveerse los componentes necesarios para la ejecución de las pruebas, su interacción con los sistemas bajo prueba y la plataforma de base.

7.1. Motivación

El mecanismo previsto por el estándar TTCN-3 para completar la implementación es la definición de un conjunto de interfaces que deberán ser implementadas para crear el ejecutable, las mismas completan los detalles no especificados por el lenguaje TTCN-3 usando alguno de los lenguajes de plataforma previstos.

Para implementar estas interfaces el especialista puede recurrir a desarrollos realizados a medida para la ocasión o utilizar bibliotecas provistas por los fabricantes de las herramientas para faciltar la implementación de dichas interfaces. Mediante alguna de estas alternativas se proveerá a la solución elementos como, adaptadores para la comunicación con el sistema bajo pruebas (System Under Test Adapters), implmentaciones de temporizadores (timers) o acceso a otras funciones de base en la plataforma, todas estas tareas requieren utilizar funcionalidades dependientes de las plataformas de base en las que se llevarán a cabo las pruebas en cada caso. Estas interfaces proveen los puntos de interacción del lenguaje con la plataforma de base, agrupando conjuntos de funciones por especialidad para conformar las diferentes interfaces que se especifican.

Las interfaces y los conceptos relacionados con ellas fueron descritos en mayor detalle en el capítulo de introducción al lenguaje TTCN-3 en el item 1.2 *Interfaces definidas en el estándar*, la

siguiente tabla muestra un resumen de las mismas.

Interfaz	**Entidades Relacionadas**
TRI triCommunication (TRI-SA)	TE / SA
TRI triCommunication (TRI-SA)	TE / SA
TRI triPlatform (TRI-PA)	TE / PA
TCI Test Management Interface (TCI-TM)	TE / TM
TCI Component Handling Interface (TCI-CH)	TE / CH
TCI Coding/Decoding Interface (TCI-CD)	TE / CD
TLI Test Logging Interface (TCI-TL)	TE / TL

Tabla 7.1: Interfaces y Entidades relacionadas

El estándar que define TTCN-3 especifica dos lenguajes para el mapeo de estas interfaces a lenguajes de plataforma, un mapeo para C/C++ y otro para Java. Las herramientas y compiladores de TTCN-3 en su estado del arte al momento de realizar este trabajo se habían enfocado principalmente en la especialización de una de las plataformas, ignorando en general la otra. Este proceso se dio naturalmente por la especialización de los equipos de desarrollo, e incluso las empresas, en una de las plataformas y la ausencia de soluciones que ofrecieran mecanismos de interoperabilidad a costos razonables. Algunos casos que validan esta afirmación son los productos TTWorkBench de Testingtech [39] que originalmente se espacializó en la plataforma Java y Exhaustif/TTCN de MTP o Tester de Telelogic que se han especializado en la plataforma C/C++, todos ellos de peso relativamente importante en la industria.

Para la construcción de los ejecutables, muchas de las soluciones existentes en la plataforma C/C++, traducen los ATS a uno de los dos lenguajes de plataforma, generando el llamado ATS compilado (Compiled ATS - C-ATS) [22]. Luego este C-ATS es enlazado con las implementaciones de TCI y TRI provistas en cada caso, generando así el correspondiente ejecutable ETS. De esta forma para dichos casos, esta decisión tecnológica genera incompatibilidad con la otra plataforma, al limitar las capacidades de enlazado a implementaciones en la misma plataforma del compilador. Algo equivalente ocurre cuando una herramienta opta por la plataforma Java y no tiene la capacidad de ejecutar de forma sencilla componentes desarrollados en C/C++.

Por estas causas se constituyó una situación *de facto* donde diferentes herramientas al optar por una plataforma no pueden acceder de forma nativa a componentes o bibliotecas desarrollados en la otra, recurriendo a soluciones de adaptación como la utilización de JNI u otras tecnologías cuando, para realizar una implementación concreta, es imprescindible cambiar de plataforma.

La situación generada tiene consecuencias importantes respecto de la portabilidad y reuso de soluciones y bibliotecas, afectando negativamente la posibilidad de una utilización más amplia del lenguaje y en particular el reuso de bibliotecas entre plataformas.

El problema enfrentado por la industria respecto de esta situación, es que solo algunos casos de prueba pueden ser implementados eficientemetne usando una de las plataformas, ya que muchas veces la naturaleza de los casos de prueba determinan que una plataforma sea más adecuada para su implementación, por tanto los Adaptadores necesarios para este tipo de pruebas se desarrollan en dicha plataforma y determinan el uso de herramientas correspondientes a la misma o la necesidad de desarrollar *wrappers* a medida para integrarlas en herramientas de otras plataformas.

Si bien, la situación *de facto* descrita cumple con la especificación del estándar para el lenguaje, esta no es la única interpretación que admite el mismo, el estándar no fuerza a elegir entre una

de las plataformas. Un enfoque alternativo puede ser un compilador que implemente en forma nativa las interfaces para ambas plataformas de forma simultánea, permitiendo así la integración con ambas de la forma más simple posible. Para esto mediante la definición de algún mecanismo en la herramienta, se podría especificar la plataforma adecuada para cada elemento del lenguaje, en los casos donde esto fuese necesario o deseable.

Para demostrar que esta nueva interpretación del estándar era posible, nos propusimos la definición, el diseño y la implementación de una prueba de concepto para ofrecer una solución con estas características, la cual debería habilitar que desde una herramienta TTCN-3 se tenga la capacidad para realizar implementaciones en ambas plataformas de base.

La nueva interpretación del estándar que propusimos cumple completamente con las especificaciones, ya que no requiere ningún cambio en este para que pueda estar operativa una solución con estas características. Lo que se requiere en cambio, para su implementación, es una modificación de las herramientas correspondientes, de forma que estas cuenten con el soporte a ambas plataformas de foma nativa y simultánea, habilitando por medio de algún mecanismo a establecer, la selección de la plataforma para cada elemento del lenguaje en una implementación concreta. En consecuencia esta solución dejaría disponibles para los desarrolladores, las bibliotecas y herramientas, que estos precisan para una implementación específica, más allá de su plataforma de base, mediante un mecanismo sencillo y con un esfuerzo razonablemente bajo.

7.2. Objetivos

Para probar que esta solución era correcta y podía aplicarse con un esfuerzo razonable, nos propusimos como prueba de concepto realizar una extensión experimental a un compilador TTCN-3 ya existente y sus herramientas relacionadas, logrando que este fuese capaz de interoperar con bibliotecas y herramientas en otras plataformas en forma nativa y con un costo razonable para el desarrollador.

Una herramienta TTCN-3 con estas características permitirá a los desarrolladores, más allá de la plataforma de base de la herramienta seleccionada por ellos, integrar bibliotecas y herramientas desarrolladas en otras plataformas de forma sencilla y eficiente. Usando estas herramientas de testing los desarrolladores podrán recurrir a las bibliotecas y adaptadores de la plataforma más conveniente en cada caso, concentrándose en cual provee la funcionalidad más adecuada a menor costo, sin preocuparse de los costos de integración y sin modificar la programación en TTCN-3.

Para lograr esto y poderlo probar empiricamente era imprescindible disponer de un compilador, con el cual pudiéramos experimentar las soluciones propuestas, esto significaba no solo sus licencias de uso sino también el acceso a su código fuente y la capacidad de modificarlo. No existían al momento de analizar esta necesidad herramientas TTCN-3 disponibles con estas características, por lo cual se consideró la alternativa de desarrollar una herramienta abierta de TTCN-3. Esta tarea fue acometida en el contexto del proyecto Go4IT [22], el proceso para construir el compilador abierto y libre (open/free) ρTTCN-3 fue descrito en la primera parte de esta tesis “Un compilador abierto para TTCN-3: μTTCN-3 y ρTTCN-3”. La versión A0 de ρTTCN-3 quedó disponible durante el año 2008 y estableció las bases requeridas para desarrollar esta segunda etapa de nuestro trabajo.

La figura 7.1 describe la arquitectura del compilador ρTTCN-3 utilizado como punto de partida para este trabajo.

El diagrama muestra en su centro el compiled ATS (C-ATS), el cual representan el código generado a partir de TTCN-3, en nuestro caso en C/C++. En torno al C-ATS podemos observar los diferentes componentes del Run Time System (RTS), los cuales dan soporte a algunas funcionalidades necesarias del ATS y en particular brindan soporte para la ejecución de las interfaces definidas

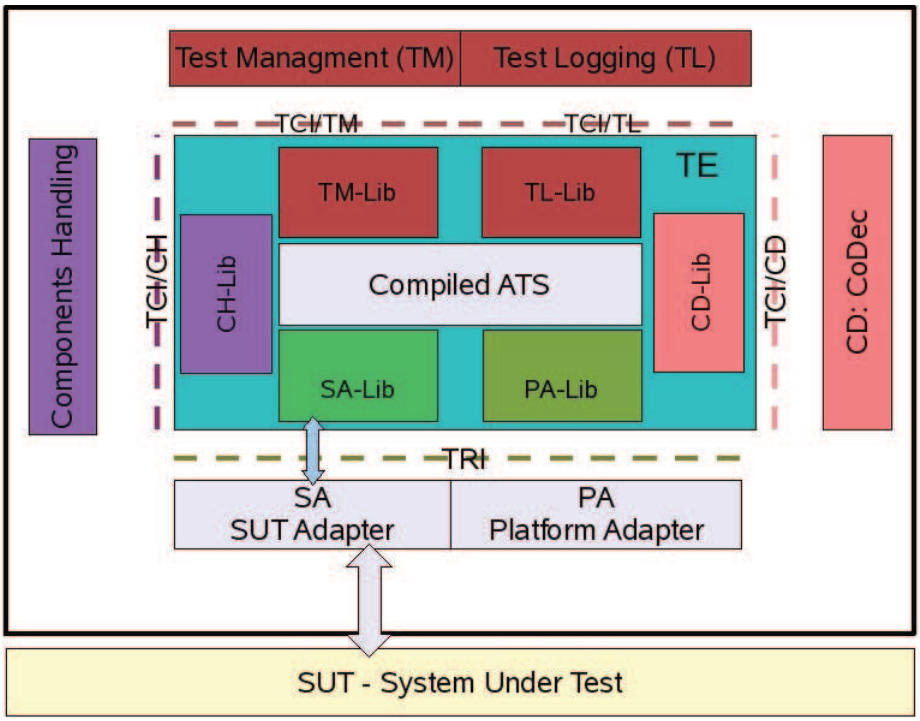

Figura 7.1: Go4IT ρTTCN-3 Arquitectura inicial del compilador

en el estándar, que pueden observarse más afuera en el diagrama. Los componentes más relevantes a efectos de nuestra prueba de concepto son los que dan soporte a las interfaces, en particular a la TRI y la TCI que es donde deberemos realizar los cambios necesarios para dar soporte a las diferentes plataformas de base. Podemos observar en el diagrama que el Compiled ATS utiliza los servicios de la SA-Lib, la PA-Lib y la CD-Lib para soportar estas interfaces, de aquí se determinarán que bibliotecas deberemos modificar en cada caso.

Capítulo 8

Antecedentes

En esta sección haremos un revisión de la experiencia acumulada en TTCN-3 en particular en nuestro entorno cercano, mencionaremos algunos trabajos presentados en las Conferencias de Usuarios de TTCN-3 (TTCN-3 User Conference - T3UC) de los últimos años. Nos concentraremos en los temas de interoperabilidad que son el centro de nuestro interés, presentaremos aquellas experiencias que hemos tenido en consideración para proponer y solucionar nuestro trabajo.

Se cuenta ya con una década de experiencia en el diseño e implementación de casos de prueba sobre la versión 3 de TTCN, partiendo de los artículos sobre el diseño del lenguaje [23, 24]. Nuestro campo de experiencia nos ha influenciado fuertemente en la implementación de soluciones con el mapeo C/C++ del estándar TTCN-3. Contribuciones al T3DevKitv [3] y al proyecto Go4IT [22], entre otras realizadas por diferentes integrantes del equipo, están en el mundo C/C++. Como instituciones relacionadas a la investigación, encontramos además un desafio en trabajar también en la implementación Java, de forma de validar la cobertura a todas las necesidades que pueden plantearse desde la industria, logrando completar las alternativas existentes desde el punto de vista tecnológico.

En las oportunidades en que se debió enfrentar la necesidad de implementar en Java conjuntos de pruebas ya existentes en las plataformas C/C++, se evaluaron como la alternativas la reimplementación de dichas soluciones en Java, utilizando una herramienta TTCN-3 en esta plataforma, o la utilización de mecanismos de interoperabilidad. Se verificó que la primera alternativa requería de un esfuerzo adicional importante, además en muchos casos, la plataforma no era adecuada para la naturaleza del problema a abordar y las dificultades para la implementación serían naturalmente mayores, incrementando el esfuerzo más allá del realizado en la plataforma original. Estas razones, fundamentadas en experiencias anteriores, nos llevaron en general a descartar la alternativa de reimplementar en la otra plataforma las soluciones ya existentes. Es así que en general preferimos trabajar en el sentido de integrar las soluciones ya desarrolladas en C/C++ a la plataforma Java, mediante el uso de las tecnolgías de interoperabilidad disponibles en cada caso, a continuación describimos algunas de estas experiencias realizadas por nuestro equipo y otras de terceros, las cuales nos orientaron al formular la presente propuesta. También presentamos algún caso de re-uso de trabajos realizados en Java desde C/C++.

8.1. Experiencias de Interoperabilidad

Algunas experiencias de interoperabilidad de plataformas, con requerimientos similares a los que nos propusimos resolver de forma genérica en este trabajo, provienen directamente del proyecto Go4IT [22], dada nuestra vinculación a éste las mismas se convirtieron en las fuentes principales para nuestro análisis. Fue en el contexto de estos proyectos que pudimos valorar en su justa medida

el esfuerzo que implicaba implementar la interoperabilidad entre plataformas para el compilador y las implementaciones o el costo de migrar una solución de herramientas para cambiar de plataforma de implementación.

Nuestro análisis se basa en dos experiencias principales, la primera consistió en habilitar el uso desde implementaciones Java del CoDec Generator(CDGEN) parte del T3DevKit [3], una de las herramientas producidas en el proyecto Go4IT. La segunda experiencia es sobre la integración de las herramientas GUI desarrolladas en Java en el contexto del Package 1 de Go4IT con el compilador ρTTCN-3 desarrollado en C/C++. En el ámbito de estos dos proyectos se requería tanto invocar código C/C++ desde Java, como código Java desde C/C++.

Se investigaron de forma complementaria otras experiencias relacionadas con interoperabilidad de plataformas desde TTCN-3 que no estaban relacionadas a Go4IT o los equipos a los que nos encontrábamos vinculados. Otros equipos habían enfrentado antes que nosotros la necesidad de usar diferentes plataformas y habían evidenciado las dificultades y costos relacionados con este proceso. Presentamos a continuación las referencias a los principales trabajos publicados como ejemplo de requerimientos de interoperabilidad entre plataformas y los mecanismos utilizados para resolverlos.

Como veremos en los ejemplos la construcción de *wrappers* es la técnica mas usual para implementar la invocación entre plataformas. En este contexto un *wrapper* es una pieza de software, que permite invocar a otras piezas de software en general preexistentes en una plataforma de base diferente de la que invoca, la idea es que el wrapper envuelve a la pieza original, presentando una interfaz para la otra pieza de software que precisa invocar a esta.

T3DevKit Wrapper for Java Este trabajo es el más cercano a nosotros en algunos aspectos, se realizó como un proyecto de grado de InCo/fing/UdelaR, su objetivo fue el reuso del T3DevKit desde entornos Java. Participamos en el mismo en forma directa en la etapa de diseño de la solución, esta será desarrollada en la sección 12.5, para profundizar en el proyecto pude referirse a su documentación [26]. El mismo consistió en extender el T3DevKit, permitiendo generar implementaciones TCI y TRI para Java, esto permite utilizar las funcionalidades de automatización que brinda el T3DevKit desde la plataforma Java, mejorando de forma relevante la portabilidad del T3DevKit en TTCN-3 para esta plataforma. Durante el proyecto se realizó una prueba de concepto sobre la herramienta TTWorkBench de Testingtech [39], modificando la misma para habilitar el uso del T3DevKit desde esta. Para la implementación de esta prueba de concepto, se abordaron aspectos tecnológicos similares a los enfrentados en este trabajo, aunque aplicados de forma más concreta, en particular la utilización de JNI (Java Native Interface) como herramienta de interoperabilidad, a nivel de código fuente, entre C/C++ y Java. Por estas razones fue el antecedente más importante en el que nos apoyamos para el análisis y la toma de decisiones de los aspectos tecnológicos en nuestro trabajo.

Testing Tools Java Wrapper for Go4IT C/C++ compiler Relacionado con nuestro trabajo de construcción del compilador ρTTCN-3 en el package 2 del proyecto Go4IT, se realizó esta implementación de interoperabilidad con el mismo. El objetivo del proyecto fue extender el uso de las herramientas de Gestión de Pruebas (Test Managment) de la herramienta TTWorkBench de Testingtech [39] al compilador C/C++ ρTTCN-3. Este proyecto de alguna forma complementa la experiencia del anterior, en los aspectos tecnológicos mencionados ya que utiliza la tecnología JNI desde la perspectiva del desarrollo de aplicaciones de usuario Java, validando de forma más completa la utilización de esta tecnología y cubriendo el resto de las interfaces definidas en el estándar.

Modeling external activities in PERL Este trabajo [38] discute los conceptos relacionados con comportamientos externos y la ejecución de funciones implmentadas en PERL desde

TTCN-3, propone extensiones a TTCN-3 para soportar estos y demostrar su aplicación en ejemplos concretos. El mismo nos sugirió la posibilidad del cruce entre múltiples lenguajes de plataforma como otra aplicación posible del concepto extendido de dualización.

Mapping to the new platform: .net La propuesta [40] trata sobre la extensión del estándar a un nuevo lenguaje, definiendo la implementación de la interfaz para C#. Esto permitiría no solo el uso de este lenguaje, sino a través de el a la Paltaforma .NET sus bibliotecas y a todos los demás lenguajes que esta maneja, siendo una alternativa al problema surgido desde el item anterior para los lenguajes de plataforma en Microsoft .NET.

La diferencia entre este enfoque y el de la dualización es que mientras este implicaría un cambio en el estándar el de la dualización reusaría los mapeos de las interfaces ya existentes.

8.2. Experiencia en JNI

Más allá de información que podemos recabar de reportes de los proyectos mencionados en la Sección 8.1, nuestro equipo de trabajo más cercano participó en algunos estos de forma directa, lo cual nos permitió adquirir cierta experiencia práctica en el área de interoperabilidad entre las plataformas de base en implementaciones TTCN-3. Una de las principales conclusiones que surge de estas experiencias es la capacidad de Java Native Interface (JNI) [36] para resolver los problemas que nos planteábamos respecto a la interoperabilidad en proyectos concretos de TTCN-3. Dados nuestros objetivos, relacionados con la Solución Dual, nos interesó profundizar en aquellas experiencias relacionadas al uso de JNI, la cual evaluamos posteriormente en 10.1, como una de las herramientas de Interoperabilidad entre las plataformas Java y C/C++ para la realización de nuestra prueba de concepto. Pudimos confirmar que JNI resuelve satisfactoriamente la ejecución entre las plataformas de acuerdo a nuestras necesidades y brinda las herramientas para el intercambio de datos u objetos entre las mismas, tanto desde Java a C/C++ como en el sentido inverso. Respecto al esfuerzo necesario para su utilización, se verificó que una vez comprendidos y resueltos algunos aspectos complejos, como la especificación de los tipos en las firmas de las funciones nativas o utilización de maquinas virtuales embebidas, se logra un nivel de eficiencia aceptable en el desarrollo.

Partiendo de estas experiencias algunas preguntas quedaban pendientes para investigar, las más relevantes eran:

Nivel de abstracción Los nivelese de abstracción que se requiere manejar en una implementación concreta son distintos respecto a la construcción de una herramienta genérica que brinde interoperabilidad de forma dinámica. Deberíamos resolver en este sentido la invocación de forma completamente dinámica a componentes de las distintas plataformas.

Plataforma de partida Mientras los proyectos conocidos se paraban en plataforma Java nuestra experiencia concreta y la prueba de conceptos que nos planteábamos partía de la plataforma C/C++, además la misma concepción de JNI está inspirada en ser una herramienta para la interoperabilidad desde la plataforma Java. Nos quedaba pendiente entonces validar la utilización de la tecnología en el otro sentido, por un lado, a diferencia de la utilización de C/C++ desde Java, aquí se precisaría la gestión del entorno de ejecución Java. En este sentido nos preocupaba especialmente como lograr que las instancias de procesos, con los datos asociados a este, creadas en el lado Java estuviesen disponibles a lo largo de toda la ejecución del compilador del lado C/C++.

Capítulo 9

Definición y Alcance de la Solución dual

El principal objetivo en el diseño del compilador dual fue la generalización de la solución de integración con la plataforma de base no nativa, buscando mantener la solución transparente para los usuarios finales de TTCN-3 y los desarrolladores de pruebas. Otro objetivo, secundario, del diseño fue proponer una idea independiente de las plataformas, de forma que la misma pudiese reusarse en otros compiladores y herramientas de la industria, ofreciendo de esta forma un camino para dualizar otras plataformas TTCN-3 ya existentes.

De acuerdo a estos objetivos, podemos decir que una solución es dual cuando cumple con la siguiente definición:

Una herramienta para el desarrollo de soluciones en TTCN-3 que, de forma transparente, permite utilizar implementaciones y bibliotecas en cualquiera de las plataformas de base, definidas en el estándar para el mapeo de sus interfaces.

9.1. Definiciones generales de la Solución Dual

Para conseguir estos objetivos evolucionamos naturalmente a la idea de factorización de las técnicas de integración, ya utilizadas en las experiencias anteriores, que nos permitiera tener una solución general de interoperabilidad, evitando de esta forma el trabajo de implementar en cada caso el cruce de plataforma. El comportamiento deseado era que, para diferentes elementos del lenguaje, como puertos, funciones externas u otros, el ejecutable decidiera sobre que plataforma ejecutar un requerimiento concreto. El aspecto decisivo para definir la solución fue determinar en que momento y lugar de la herramienta TTCN-3 debíamos incorporar la lógica para la dualización de la misma, logrando de esta forma acceder a las implementaciones en ambas plataformas de la forma más simple y conveniente posible.

9.1.1. Donde incorporar la Dualización

La Figura 7.1, que describe la arquitectura del compilador ρTTCN-3 que elegimos dualizar en nuestra prueba de concepto, nos muestra especialmente el diseño de componentes, relacionándolo con las interfaces definidas para acceder a las partes de la solución que se ejecutan en las plataformas de base, siendo precisamente la implementación de estas interfaces la que debíamos dualizar para poder acceder a ambas plataformas indistintamente.

Incorporar la solución al compilador aparece, en primera instancia, como la opción más adecuada para lograr los objetivos planteados, ya que de esta forma se coloca la mayor parte de la solución fuera del ámbito del desarrollador TTCN-3 o el responsable por la ejecución de los casos de prueba. Debíamos entonces prototipar este escenario usando el compilador experimental ρTTCN-3,

desarrollado en la primera etapa de nuestro trabajo, para lo cual lo primero era definir en que etapa del compilador era más adecuado agregar este soporte.

La primer alternativa que evaluamos fue realizar las modificaciones en la etapa de traducción, claramente esto solucionaba el problema planteado y se podía automatizar la selección de la plataforms basado en alguna especificación que se leyese en tiempo de compilación, agregando la lógica correspondiente al código C/C++ generado en la traducción. Un elemento de mucho peso en esta alternativa era que se encontraba en un área del compilador que nos resultaba más accesible y controlada, dado nuestra participación en el diseño e implementación de la misma. Sin embargo, evaluando esta alternativa desde el punto de vista del usuario o desarrollador TTCN-3, nos dimos cuenta que la misma era **estática en tiempo de ejecución**, es decír que al tener que recorrer la etapa de traducción para aplicar un cambio, debía compilarse o empaquetarse la solución nuevamente según la plataforma de base que se usara, para que el cambio se hiciese efectivo durante la ejecución. Esto tenía como consecuencia que una vez compilado el ETS no podía cambiarse la plataforma de un elemento sin regenenerar el TE, con la consecuente perdida de tiempo y flexibilidad. Cada vez que se requiriese un cambio de la plataforma de algún elemento el ETS debería regenerarse, actividad que usualmente es demasiado costosa en tiempo en las diferentes herramientas de TTCN-3.

Teniendo esta consideración en cuenta evaluamos la alternativa de ubicar los cambios relacionados a la dualización en el Sistema de Tiempo de Ejecución (RTS), verificando que esta alternativa presentaba ventajas en otros aspectos:

Simple de usar, es la que simplifica más la solución a los usuarios y desarrolladores de TTCN-3, ya que permite concentrar las definiciones sobre las plataformas de los elementos en un único punto de especificación, por ejemplo un archivo u otro tipo de registro.

Estrategia genérica, desde el punto de vista de la dualización de la plataforma, identifica claramente un lugar de la misma donde realizar los cambios, este punto no es opcional para una implementación concreta de TTCN-3 ya que, hasta cierto punto, está definida en el estándar, lo cual brinda una estrategia posible para extender la idea de dualización a otras herramientas.

Dinámica, lo más importante es que esta opción es la más dinámica de todas al permitir cambiar la plataforma de un elemento en tiempo de ejecución, simplemente cambiando la configuración del mismo y volviendo a ejecutar la prueba.

Por estos motivos esta opción es la que encontramos más adecuada para nuestros objetivos, si bien tenía para nosotros una dificultad extra: nuestra falta de conocimiento del RTS, su diseño e implementación.

Al ubicar la lógica para la dualización en el RTS, cuando en tiempo de ejecución el TE realiza las llamadas a los diferentes componentes mediante las interfaces definidas en el estándar, la biblioteca del RTS que implementa la misma deberá conocer la plataforma de base del elemento correspondiente y entonces lo ejecutará a través de la implementación correspondiente de la interfaz, la cual deberá estar disponible para ambas plataformas, las que pordrían ser Java o C/C++.

Según las especificaciones del estándar, la inicialización de estos componentes debe realizarse cada vez que se ejecuta una Suite de Pruebas, lo cual garantiza el grado de dinamismo requerido, aunque dependiendo del diseõ de la herramienta este podría además resetearse a diferentes niveles durante la misma ejecución.

En el otro sentido, desde la visión de las herramientas para la gestión y el monitoreo del compilador, la dualización de las interfaces correspondientes permitiría a dichas herramientas trabajar con compiladores en ambas plataformas mejorando así también la portabilidad de las mismas.

9.1.2. Abstracción de la Plataforma y Portabilidad

Con estas definiciones avanzamos un paso en la capacidad de abstracción y la portabilidad de las implementaciones en TTCN-3, al abstraer también la plataforma en la cual se ejecutan los componentes de la solución que se implementan fuera del lenguaje TTCN-3. Cuando otra plataforma va a ser usada para implementar algún componente, el desarrollador solo necesitará agregar la configuración correspondiente al elemento, su plataforma y la información complementaria necesaria para localizar y cargar la implementación de dicho elemento. El desarrollador TTCN-3 no tiene que dedicar esfuerzo en resolver como integrarse, desde un punto de vista técnico, con una u otra plataforma, siempre realiza su tarea de la misma manera, delegando completamente el desarrollo de los adaptadores al equipo especializado en la plataforma correspondiente.

9.1.3. Adhesion al estándar TTCN-3

Otro de los objetivos planteados en el diseño de la solución fue cumplir estrictamente con las especificaciones del estándar que define el lenguaje, para esto debíamos mantener la implementación de las interfaces de TTCN-3 de acuerdo a las definiciones dadas en el mismo.

Para lograr este objetivo nos propusimos aislar de la implementación el soporte para la dualización, en un único modulo especializado para esto, para luego implementar las interfaces con cada plataforma separadamente de acuerdo al estándar, las cuales serían luego invocadas en cada caso de acuerdo a la semántica operacional en cada caso. De esta forma la complejidad e inteligencia necesarias para resolver la invocación a la plataforma correspondiente, de acuerdo a la especificación de cada elemento, se localizarían en algunas de las bibliotecas que forman parte del RTS, para a partir de allí invocar a la interfaz correspondiente, cumpliendo de esta forma con lo especificado en el estándar.

Este cuidado en la definción del lugar y forma de introducir estos cambios no responde únicamente a un afan purista de cumplir con el estándar, sino que el cumplir con este tiene consecuencias prácticas ya que nos permite mantener los niveles de interoperabilidad perseguidos por el mismo al definir estas interfaces.

Por otro lado, se realizó un esfuerzo para minimizar la información agregada y los cambios realizados al T3RTS, esta información representa la configuración de la plataforma para los objetos de TTCN-3 que puedan estar implementados en estas, dicha información será utilizada por la lógica incorporada al RTS para tomar la decisión de cual interfaz invocar al momento de utilizar cada objeto en particular.

9.2. Alcance de la solución

Para nuestros objetivos específicos, de validar la Solución Dual, encontramos suficiente poder implementar la dualización para algunos elementos del lenguaje a efectos de probar paracticamente la solución propuesta. Entendimos que los elementos más interesantes a tener en cuenta para nuestra implementación eran **puertos**, **temporizadores**, **codificadores** y **funciones externas**, ya que los mismos son los elementos de las soluciones TTCN-3 implementados en los lenguajes de plataforma por los desarrolladores.

Para acotar el trabajo durante la prueba de conceptos optamos por implementar la dualización de los puertos, estos juegan un rol fundamental en la problemática que nos planteamos atacar ya que son necesarios para comunicarse con el SUT y por lo tanto son los más expuestos a requerir una plataforma diferente a la de la herramienta. También en nuestra opinión estos son el elemento más representativo de las abstracciones para pruebas en TTCN-3 para los cuales disponíamos de

los casos de prueba para el test de regresión. Más allá de esto para poder utilizar los puertos en la otra plataforma debíamos contar soporte dual para el intercambio de datos entre TTCN-3 y el adaptodor en la otra plataforma.

De acuerdo a estas definiciones, la herramienta una vez dualizada debería seleccionar la plataforma correspondiente, en tiempo de ejecución, sobre la cual ejecutar cada elemento, sin necesidad de realizar ninguna actividad específica de programación por parte de los desarrolladores TTCN-3.

Según nuestra propuesta, para lograr este comportamiento la solución solo requeriría cierta configuración adicional, donde se especificarían los elementos con su plataforma correspondiente, este sería el unico trabajo extra necesario por parte de los desarrolladores e implementadores de TTCN-3.

La información a especificar debería ser Tipo de Elemento, Identificador del Elemento y su Plataforma, donde:

- Tipo de elemento, representa al tipo de estructura del lenguaje a que nos referimos como Puertos, Temporizadores, Funciones Externas o CoDecs,
- Identificador de Elemento, es el nombre o identificador que se le puso a la instancia del elemento al programar en TTCN-3
- Plataforma, es una de las posibles en la Solución Dual, para el alcance actual de TTCN-3 C/C++ o Java.

Esta especificación es consistente y completa pues, de acuerdo a las definiciones del estándar, los identificadores de estos elementos deben ser únicos en una Suite de Pruebas determinada, lo cual garantiza la identificación de cada elemento a especificar dentro de una implementación. La presencia del Tipo de Elemento, si bien es obligatoria, cumple un rol organizativo para ordenar el trabajo de los implementadores pero no es requerido a los efectos de la identificación del elemento.

Para mantener la compatibilidad con la solución sin dualizar, de la que partimos, decidimos tener como comportamiento por defecto el de la plataforma de base de la herramienta a la que se está incorporando comportamiento dual, de esta forma todo elemento que no estuviera especificado explícitamente se consideraría implementado en la plataforma original y sería ejecutada por el mecanismo nativo de la plataforma TTCN-3. Para nuestro caso en el que incroporaremos soporte dual a ρTTCN-3, cuya plataforma de base es C/C++, cuando un puerto, u otro elemento del lenguaje que se implementa en una plataforma de base, no esté específicado en el archivo de configuración se asumirá como implementado en C/C++, en el caso de ρTTCN-3 esta implmentación se proveerá a través del T3DevKit.

Para la implementación de esta configuración evaluamos algunas alternativas como variables de entorno, archivos con formato propietario o archivos XML. Las variables de entorno podían depender de la plataforma subyacente, lo cual no las hacía demasiado ventajosas y un archivo propietaria agregaba complejidad y requería mayor conocimiento específico de la solución por parte del desarrollador. Finalmente entendimos que la mejor opción para la implementación de esta especificación, era utilizar archivos en formato XML, lo cual nos dejaría mayores capacidades de expresividad para futuras extensiones y una forma simple de manejo para el implementador TTCN-3.

Capítulo 10

Aspectos Tecnológicos

Abordaremos en esta sección algunos aspectos de tecnológicos de relevancia para la Solución Dual, el principal desafío en este aspecto lo presento la selección y el uso de la tecnología de interoperabildad a usar en el desarrollo de la misma, en consecuencia nos concentraremos en estos dos puntos.

10.1. Selección de la tecnología

Más allá de que nuestros conocimientos y experiencia concreta nos inclinaban el uso de JNI para la resolver la interoperabilidad entre C y Java, nos planteamos evaluar otras alternativas tecnológicas para lograr interoperabilidad entre ambas plataformas.

La mayor parte de las otras tecnologías consideradas para implementar la interoperabilidad, fueron dejadas de lado por estar orientadas a integración a nivel de servicios o componentes y no de código fuente. Este tipo de arquitecturas no eran adecuadas a la opción tomada de introducir la dualización en el RTS de acuerdo a los descrito en 9.1, no nos pareció adecuado para un proceso de compilación y enlace para producir un ejecutable usar una arquitectura de este tipo, que generaría necesidades de desplegar otros componentes ajenos a la solución así como posibles puntos de falla completamente ajenos a la naturaleza del problema que se trataba. En esta categoría podemos destacar tecnologías como **Servicios Web** o diferente tipo mensajería entre componentes como comunicación por **sockets** o **pipes**.

Respecto a otras tecnologías orientados a código fuente al igual que JNI, si bien parecían en general ser menos costosas de utilizar que JNI, contaban con implementaciones parciales y no cubrían los requerimientos de generalidad perseguidos por nosotros, en esta categoría podemos citar JNA [37].

Como contrapartida muchas soluciones específicas fueran desarrolladas, utilizando tecnología JNI, por diferentes proveedores basados en ambas plataformas para integrar trabajo de la otra, más aún como establecimos en 8.2 varios equipos, cuyos integrantes estaban relacionados a nuestra actividad, habían experimentado con ella.

Además del resultado concreto de la utilización de JNI en estas experiencias, nos resultó importante la existencia de documentación y experiencia suficientes en la industria como para considerarla una de nuestras opciones más firmes.

De acuerdo a lo expuesto en 8.2, considerando el resultado de dichas experiencias algunos temas quedaban aún pendientes de dilucidar en la práctica antes de tomar una decisión, las respuesta encontrads a estos temas fueron:

Nivel de abstracción Los niveles de abstracción que se requieren manejar en una implementación

concreta son distintos respecto a la construcción de una herramienta genérica que brinde interoperabilidad para cualquier elemento del lenguaje. La necesidad de utilizar dinámicamente, de acuerdo a cierta configuración que se leerá en tiempo de ejecución, algunas características de JNI, requiere la capacidad de invocar dinámicamente dichas funcionalidades, en cambio estos aspectos para una implemetación concreta se resuelven explícitamente en el código fuente y quedan establecidos en tiempo de compilación. Este aspecto se respondió afirmativamente en base a la capacidad de utilizar referencias pasadas como variables que soportaban las primitivas de JNI, característica claramente expuesta en la documentación correspondiente de JNI **The Java (tm) Native Interface Programmer's Guide and Specification** [36], esto permite que estas referencias sean resueltas en tiempo de ejecución mediante enlaces dinámicos. En dicha documentación, también se explica la posibilidad de realizar el registro de funciones en forma dinámica mediante el uso de la función *RegisterNatives()*, lo cual habilita la utilización de funciones de la aplicación llamadora o de bibliotecas dinámicas, así como un grado más de dinámismo en la carga de las definiciones de las funciones.

Plataforma de partida Basados en nuestra experiencia directa y en la de los demás trabajos tomados como referencia, no contábamos con evidencia concreta de la utilización extensiva de JNI para trabajar desde aplicaciones desarrolladas en C/C++ con componentes desarrollados en Java, al tratar de buscar referencias en Internet encontramos algunos comentarios de desarrolladores donde se cuestionaba este camino por difícil y oscuro, debido a esto pusimos especial interés en investigar sobre este aspecto. Este punto nos preocupó de manera particular y logramos dilucidarlo en forma teórica recurriendo nuevamente a la documentación ya citada de JNI en su capítulo **7 The Invocation Interface**, para la toma de decisiones, pero quedó pendiente su comprobación práctica durante la ejecución del proyecto. Una referencia, muy tranquilizadora en este sentido, fue la utilización de JNI desde los exploradores Web donde se realiza un uso extensivo de esta tecnología para ejecutar aplicaciones Java desde dichos exploradores que algunos de los cualese se encuentran escritos en C/C++.

Por otro lado de acuerdo a la documentación de JNI mencionada antes, en el punto **1.3 Implications of using the JNI**, la utilización de JNI desde la perspectiva de una solución Java, conlleva algunos riesgos que deberían tenerse en cuenta al momento de su utilización.

Pérdida de Portabilidad Al usar JNI puede haber una pérdida de portabilidad de la aplicación, desde la perspectiva de Java, ya que al acceder a recursos de la plataforma de base estos no serán necesariamente compatibles entre diferentes plataformas de base. En nuestro caso este problema no es de recibo, ya que es intrínseco a la situación a resolver y a la solución que estamos proponiendo acceder a las plataformas de base, puesto que estamos resolviendo un problema de interoperabilidad para ciertas plataformas de base específicas y nos vemos obligados a salir de la máquina virtual Java. Para atenuar esta situación intrínseca a nuestra solución realizamos un esfuerzo por lograr que la misma tenga características de portabilidad acordes a su función, este punto lo desarrollamos más adelante en la discusión del diseño de la solución en 11.

Tipos Seguros El otro aspecto a tener en cuenta es que al salir del entorno de tipos seguros de un lenguaje como Java hacia entornos de tipos más tradicionales, como es C/C++, se pierden estas condiciones y se expone la solución al riesgo de errores por corrupción de datos, la forma de minimizar este riesgo es centralizar el manejo de la interacción mediante JNI en pocos componentes para tener un mayor grado de control de la misma. Este aspecto ya se había tenído en consideración en nuestras pautas de diseño como quedó establecido en 9.1.

Siguiendo el análisis de la documentación mencionada y de acuerdo a lo expuesto en esta en el punto **1.4 When to use the JNI**, confirmamos los criterios que utilizamos al adoptar JNI como solución de interoperabilidad, allí se destaca que cuando se requiere interoperar dentro del mismo proceso JNI es la alternativa más adecuada, a pesar del costo que puede significar su uso.

Por los motivos expuestos la tecnología JNI resultó ser más adecuada a nuestros objetivos, siendo además una solución de interoperabilidad orientada al código fuente ya validada ampliamente por la industría. Seleccionamos entonces la tecnología JNI [36] para resolver la interoperabilidad entre las plataformas en la Solución Dual.

10.2. Conceptos y técnicas aplicados

En la mayor parte de las experiencias de interoperabilidad analizadas, la idea implementada usando JNI fué la construcción de algún tipo de *wrapper*, entendiendo por este una pieza de software que encapsula la ejecución de código en otra tecnología. En este caso dicho *wrapper* actúa como un intermediario entre ambas plataformas de base, concentrando el conocimiento sobre como comunicarse con cada una de ellas, en particular sobre los mecanismos brindados por JNI para resolver esta comunicación.

De forma muy simplificada lo que JNI nos ofrece es la capacidad de ejecutar código Java desde aplicaciones C/C++ y en el otro sentido ejecutar código C/C++ desde soluciones Java. La forma de realizar esto es que JNI permite ejecutar funciones o métodos C/C++ desde clases de Java y métodos de Java desde módulos o clases en C/C++. Mientras en el primer caso esto se realiza mediante el uso de una JVM (Java Virtual Machine) embebida en la aplicación C/C++, en el segundo caso se utilizan un conjunto de técnicas de *wrappeo* y modificadores que dejan las funciones de C/C++ visibles para las bibliotecas de soporte de JNI.

Cuando se realiza una llamada desde Java a una función nativa, el código de soporte de JNI, que escribe el desarrollador en la función que envuelve el código nativo, convierte los objetos de Java en instancias de tipos de datos simples o estructurados en C/C++, estos datos son usados para proveer a las funciones nativas los parámetros necesarios en cada caso. En el otro sentido, este código da soporte para cargar los valores de retorno en los propios objetos Java que están disponibles a tales efectos mediante un conjunto de funciones provistas por JNI, estos objetos quedaran actualizados en el ámbito de Java completando así el ciclo. Para realizar estas operaciones JNI brinda una conversión automática para los tipos primitivos y un conjunto de funciones para tratar objetos, clases y arreglos, incluyendo un conjunto de funciones especiales para la conversión de cadenas de caracters (String) a cadenas nativas.

Cuando se trata de realizar una llamada desde C/C++ a Java, los tipos de datos de C/C++ también son convertidos por el código de soporte para JNI en objetos Java que quedarán disponibles para ser usados en las llamadas a los métodos nativos Java. Para dar soporte a la instanciación de objetos y ejecución de métodos en Java mediante JNI, esta ofrece la capacidad de crear y usar una Java Virtual Machine (JVM), la cual queda embebida en la aplicación que invoca, donde estos se ejecutaran, a su vez mediante la utilización de las funciones para manipular objetos y métodos pueden invocarse constructores para la creación de objetos en la JVM creada y posteriormente ejecutar sus métodos. El manejo de la JVM desde JNI brinda un alto nivel de control y puede realizarse a través de interfaces muy flexibles, aunque a veces resulta complejo su utilización, dentro de los aspectos avanzados de JNI los más relevantes para nosotros fueron el manejo de múltiples hilos de ejecución (threads) y la capacidad de regsitrar dinámicamente funciones mediante el uso de *RegisterNatives()*. En definitiva mediante la utilización de JNI se logra ejecutar código enre ambas plataformas, transformando los tipos de datos previstos por TTCN-3 para el intercambio de datos

con las plataformas a través de las interfaces, en particular el tipo *bitstring* es extensamente usado para este intercambio.

JNI está considerada en la industria como la herramienta más completa para la interoperabilidad entre C/C++ y Java, para nuestro caso además cubre todas las necesidades que requería la solución propuesta para el soporte dual a ρTTCN-3. Para profundizar en cualquiera de los aspectos de JNI puede recurrirse a la documentación oficial **The Java (tm) Native Interface Programmer's Guide and Specification** [36]. Para profundizar en los mecanismos de JNI utilizados en nuestra implementación puede recurrirse a la documentaciń del proyecto **Generación Automática de CODECS para TTCN-3 / Java** [26].

Capítulo 11

Diseño de Componentes

Dadas las definiciones tomadas en 9.1, para llevar a cabo la implementación de la plataforma Dual debíamos realizar una adaptación del API que constituye el Sistema de Tiempo de Ejecución (Run Time System - RTS) de ρTTCN-3, a la que llamaríamos API Dual o RTS Dual. En la Figura 7.1 se muestra el diseño de componentes del RTS, el cual se explica al pie de la misma. Según puede observarse en este diagrama, para darle la capacidad a la herramienta de acceder a ambas plataformas, se requiere agregar la lógica para elegir estas en cada una de las bibliotecas que implementan las interfaces del estándar, las que están definidas específicamente para el acceso a la plataforma de base.

Además de la necesidad de modificar cada una de estas partes de las bibliotecas que conforman el RTS, definimos realizar un esfuerzo por agrupar el conjunto de funciones y servicios comunes que puediesen abstraerce para: facilitar el agregado de esta lógica, minimizar la introducción de puntos de fallos y realizar de forma más sencilla y clara las modificaciones de las bibliotecas. Generar una abstracción de este tipo permitiría además, cumplir con la recomendación de la documentación de JNI, acerca de concentrar el manejo de datos entre las plataformas para aislar así el intercambio de datos entre tipos seguros y no seguros de Java y el lenguaje de plataforma respectivamente, como desarrollamos en 10.1.

Por estos motivos, el componente principal de la API Dual es una clase independiente que implementa de forma genérica estas funciones y servicios, los cuales dan soporte a la ejecución de componentes en la otra plataforma y garantizan un tratamiento homogeno de las conversiones de datos.

En nuestro prototipo partíamos de una plataforma C/C++, como es ρTTCN-3, y debíamos brindar soporte para utilizar elementos u objetos implementados en Java. En este contexto la clase mencionada estaría localizada en una de las bibliotecas que forman el RTS, la cual es luego enlazada con otras partes del compilador para formar el TE, es desde esta clase, y solo desde esta, que se realizara la comunicación con Java usando JNI.

Basados en los servicios provistos por esta clase se debían realizar cambios específicos en ciertos puntos de las bibliotecas que forman el RTS, para poder realizar la llamada a cada interfaz de una u otra plataforma, estos cambios están ubicados en las bibliotecas que implementan las interfaces de TTCN-3, estas son **SA-Lib**, **PA-Lib**, **CD-Lib**, **CH-Lib**, **TM-Lib** y **TL-Lib**.

11.1. Clase Dual

Esta clase fue diseñada de forma que brinde los servicios y funcionalidades necesarias para la implementación de la dualización, especialmente las que pudieran aislarse del código fuente

original del RTS de ρTTCN-3. En particular aquí se encapsuló la utilización de todos los elementos relacionados con JNI, de forma de aislarlos en un único punto de la solución.

En primer término, en este módulo se realizarán todas las definiciones de tipos de datos necesarios para la dualización. Inmediatamente se localiza la inicialización de las configuraciones, por ejemplo aquellas relacionadas con la plataforma de los diferentes elementos del lenguaje, tanto su recuperación como la política de almacenamiento intermedio (buffer), que se quisiera implementar.

La función principal de este módulo es contener las interfaces intermedias, internas al RTS, mediante las cuales se realiza la llamada posterior al mapeo de la interfaz definida para acceder a la otra plataforma, la cual implementa fielmente la definición del estándar para esta.

Para poder realizar estas tareas en este módulo debimos además incluir ciertos servicios responsables de la gestión de los recursos necesarios para la implementación y el uso de JNI, los cuales terminaron incidiendo en cierta medida en el diseño del mismo.

Uno de los aspectos que se debió abordar en este sentido fue el soporte para múltiples hilos (multithreading) de los servicios brindados, requerido por la naturaleza de algunas abstracciones de TTCN-3, como el caracter asincrónico del intercambio de mensajes en los puertos o el funcionamiento también asíncrono de los temporizadores y sus señalizaciones. En relación con esto, fue necesario resolver la gestión del acceso a la JVM (Java Virtual Machine) a utilizar en el proceso, para lo cual se requería crear y mantener la instanica de la JVM, ofreciendo sus servicios desde un único hilo de ejecución al conjunto del proceso.

Estos aspectos requerían ofrecer los servicios mencionados en una arquitectura de múltiples hilos, para esto además de implementar las funcionalidades necesarias, estas se publicaron como servicios. Los detalles tecnológicos y de la implementaión de este módulo se desarrollan en la sección 5.1.

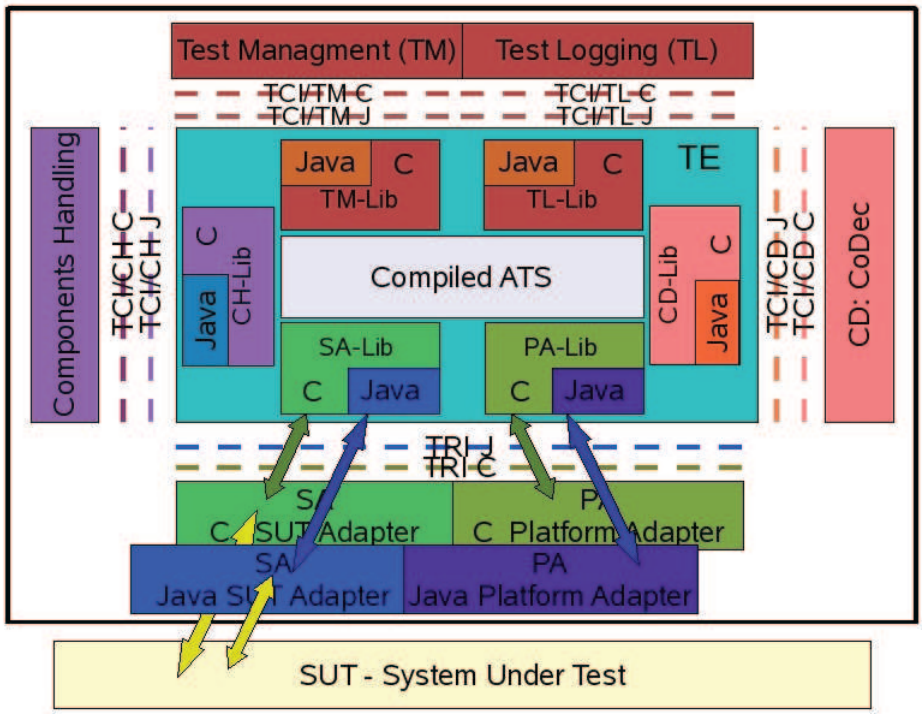

Figura 11.1: Dual API - Diseño y Arquitectura de Integración de TTCN3 dual

11.2. Cambios en las bibliotecas del RTS

Durante la construcción de ρTTCN-3, el módulo del sistema de tiempo de ejecución o RTS fue diseñado con el objetivo de aislar las tareas de tiempo de ejecución del resto de la solución, debido a los requerimientos del RTS para el tiempo de despliegue y ejecución este debe encontrarse perfectamente encapsulado, para proveer sus servicios tanto al código TTCN-3 compilado (Compiled ATS / C-ATS) como a los Adaptadores y otros componentes de la plataforma de base, mediante las interfaces definidas en el estándar. Al mismo tiempo el diseño interno del RTS trató de reflejar lo más fielmente posible los diferentes componentes o abstracciones que el estándar usa para la definición del lenguaje y su comportamiento. Esto condujo a un significativo encapsulamiento de cada una de las partes de la solución, quedando la integración entre estas delegadas en las interfaces, tanto las definidas por el estándar en caso de corresponder o a interfaces definidas internamente para aquellos casos en que el estándar no definía las mismas.

Este nivel de encapsulamiento y aislación de cada componente en el diseño del RTS, facilitó nuestro objetivo de localizar claramente los cambios a realizar para la implementación del compilador dual, a pesar de nuestro desconocimiento del mismo.

Ante este diseño orientado a objetos del RTS nos planteamos, como primera hipótesis de trabajo, modificar el diseño de los objetos del RTS para que este reflejara la capacidad dual de la plataforma. Entendemos que esta es la opción más adecuada desde la perspectiva de ofrecer una solución definitiva al problema. En contrapartida nos enfrentábamos a la falta de conocimiento detallado del diseño del RTS y cierta escasez de recursos para enfrentar un proceso de estudio y análisis del mismo que nos habilitara a realizar la tarea de este modo. Evaluando como incorporar los cambios y la información de la plataforma al RTS Dual, nos encontramos que era necesario cambiar prácticamente toda la jerarquía de objetos del mismo para introducir la información necesaria y especializar los métodos correspondientes, esto además del esfuerzo de programación que requería, agregaba una dificultad extra en la liberación de la solució dual, sus requerimientos de testing y de instalación. Considerando que se trataba de realizar una prueba de conceptos que validase la idea propuesta y no una solución definitiva, evaluamos la alternativa de modificar la lógica previamente a la invocación de las interfaces, esto podía resolverse ubicando aquellos lugares donde se invocaba cada interfaz, sin necesidad de un conocimiento profundo del diseño del RTS. Esta solución además de resolver el problema, garantizaba la completitud de la solución al intervenir en los pasos previos al uso de los componentes provistos por los lenguajes de plataforma.

Con esta opción tomada y ante un diseño donde cada componente definido en el estaándar estaba abstraído en una biblioteca específica, que a su vez implementaba cada una de las interfaces requeridas o provistas por el lenguaje, fue relativamente sencillo decidír donde localizar los cambios necesarios para la dualización. La adopción de esta alternativa fue acompañada por el diseño de una estructura, utilizada para representar la información de la plataforma de cada uno de los elementos del lenguaje, que participan en una implementación concreta y pueden ser dualizados.

En la Figura 11.1 se muestran los componentes que fue necesario modificar a efectos de obtener una plataforma TTCN-3 dualizada, a partir de la arquitectura de componentes original, ya descrita en la Figura 7.1.

Podemos observar en el diagrama como cada uno de los moódulos de las biblitoecas que soportan las interfaces debe ser modificado agregándole soporte para detectar y llamar las interfaces en Java, estos son **SA-Lib**, **PA-Lib**, **CD-Lib**, **CH-Lib**, **TM-Lib** y **TL-Lib**.

Capítulo 12

Implementación del Prototipo

Para validar las definiciones de alcance que realizamos en 9.2, con el objetivo de mostrar la factibilidad de la propuesta de la plataforma TTCN-3 Dual, nos propusimos implementar el soporte dual para algunas áreas de ρTTCN-3. Los elementos que establecimos para tener cuenta en la implementación del prototipo son **ports**, `timers`, `CoDecs` y `external functions`, debido a su característica común de tener que implementarse en las distintas plataformas de base y por lo tanto ser potencialmente objetivos de la dualización. Para ajustar el esfuerzo a los recursos disponibles, decidimos, dentro este alcance ya definido, implementar el soporte dual para los puertos y sus operaciones asociadas, esto implicaba también la implmentación del soporte dual a una parte importante de la conversión de los tipos de datos de TTCN-3.

Con este alcance podrían generarse implementaciones con puertos en ambas plataformas e incluso cambiar la plataforma de un puerto entre dos ejecuciones, de forma dinámica, sin necesidad de recompilar el ejecutable de pruebas (Test Executable - TE), simplemente cambiando la configuración correspondiente y reiniciando el sistema de pruebas, uno de los objetivos secundarios de la solución propuesta.

Veremos a en las siguientes secciones algunos aspéctos específicos de la implementación, primero nos detendremos en las modificaciones introducidas a las bibliotecas del RTS y para luego describir diferentes aspectos de la implementación relacionados a JNI. Las modificaciones que describiremos se realizaron al RTS de la versión 1.0 de ρTTCN-3, para conocer más en detalle la implementació realizada, los problemas enfrentádos y las soluciones encontradas para estos referirse al 4 15.2.

12.1. La clase T3RTSDual

Siguiendo las decisiones de diseño expuestas para nuestra implementación de prueba en la Sección 11, concentramos todas las tareas posibles relacionadas a la interoperabilidad de las plataformas en una clase con este cometido, respetando las pautas de denominación de Go4IT para ρTTCN-3 nombramos dicha clase como `T3RTSDual`. Esta clase C++ que es parte de ρTTCN-3, implementa la definición, inicialización, carga del ambiente y ejecución en la otra plataforma, en nuestro caso Java.

Es decir que tendremos esta clase en C++ que implementará la invocación de los diferentes servicios provistos por JNI para la ejecución de implementaciones Java desde la aplicación C++, en nuestro caso el RTS de ρTTCN-3.

Otra de las responsabilidades de esta clase es la recuperación y gestión de la configuración de las definiciones relacionados con la plataforma de cada uno de los elementos dualizables de una implementación concreta.

De esta forma una de las tareas más importantes abstraídas en este modulo es la utilización de JNI y el ocultamiento de su complejidad al resto de la solución, dejaremos los aspectos de la implementación relacionados con JNI para el final de esta Sección, en la Subsección 12.5, a continuación veremos en detalle los otros aspectos que mencionamos.

Definiciones Las definiciones que se implementaron en el `T3RTSDual` se refieren a enumerados para representar la información utilizada en la dualización, como los tipos de los elementos a dualizar y las plataformas en que estos pueden ser implementados.

Configuración de la plataforma Cierta información de configuración sobre la otra plataforma, necesaria para dejar disponibles los ambientes de ejecución de esta, requiere ser cargada al momento de iniciar el TE, esta información también es inicializada en la clase `T3RTSDual`. En nuestro caso para ρTTCN-3, que se encuentra en C/C++, se requiere la información para localizar el *classpath* de java y los archivos *jar* que contienen la las clases java, que forman parte de la implementación de los adaptadores y funciones a ejecutar en la otra plataforma.

Alcance de las referencias a la otra plataforma Durante el proceso de inicialización de la plataforma algunas referencias son creadas, estas deben quedar disponibles en el RTS para ser usadas durante la ejecución de los casos de prueba, en nuestro caso nos referimos a las referencias a la JVM (Java Virtual Machine) creada por JNI y a los objetos creados en la misma que contienen las implementaciones en la plataforma cruzada, estos elementos también son creados en la clase `T3RTSDual` y quedan disponibles para el resto del RTS mediante variables globales.

Plataforma de los elementos a dualizar Otra información relevante, que debe ser gestionada durante la ejecución de los casos de prueba, es la referente a la plataforma de cada elemento del lenguaje, esta tarea también se resuelve en esta clase, donde serán cargados y almacenados los elementos y su plataforma en estructuras de datos en memoria, para su posterior uso durante la ejecución de los casos de prueba.

Esta información es recuperada al inicializar el módulo desde archivos de texto diseñados a tales efectos, para ofrecer esta información internamente el módulo `T3RTSDual` provee un método que devuelve la plataforma de un elemento dado por su tipo e identificador o nombre.

Actualización Dinámica Para cumplir con los requerimientos de actualización dinámica de los elementos de la plataforma que nos planteamos, las actividades de inicialización mencionadas en el punto anterior, deben realizarse cada vez que el RTS es inicializado, si bien el estándar da algunas pautas al respecto de cuando esto debiese suceder en sus documentos, “Methods for Testing and Specification (MTS), The Testing and Test Control Notation version 3, Part 4: TTCN-3 Operational Semantics The evaluation procedure for a TTCN-3 module, Evaluation phases, Phase I: Initialization” pág 48 y “Part 1: TTCN-3 Core Language, Module control part” pág 154, el detalle de cuando este proceso es realmente ejecutado dependiendo de las opciones tomadas en la implementación de cada una de las herramientas. En el caso de la dualización del ρTTCN-3 estas tareas fueron localizadas en la ejecución del Testcase en la clase `TCICHRequired` en el método `tciExecuteTestCase`, esto nos garantiza que se inicialice cada vez que se va a ejecutar el caso de prueba, lo cual es suficiente dentro del alcance de la implementación de nuestra prueba de concepto para ρTTCN-3. Para dualizar otra herramienta TTCN-3, especialmente una herramienta de la industria, deberá evaluarse cuidadosamente cual es el lugar más adecuado para la inicialización de esta clase, buscando un compromiso entre lograr la actualización dinámica y el desempeõ de la solución, siempre dentro de la biblioteca que implementa el RTS.

12.2. Cambios en las interfaces con la plataforma

Si bien siguiendo nuestros objetivos de diseño aislamos los cambios para implementar la Solución Dual en la clase `T3RTSDual`, cierta semántica operacional de la misma está natural e intrínsecamente asociada a cada elemento del lenguaje que puede tener una implementación dual y sus operaciones, es decir aquellas operaciones del lenguaje donde existe la posibilidad de optar por una u otra plataforma. Al mismo tiempo estas operaciones están abstraídas en las interfaces previstas por el estándar a efectos de acceder a la implementación en el lenguaje de plataforma 1.2, por lo cual debíamos modificar dichos elementos en el momento de invocación a las mismas para incorporar la capacidad dual.

Desde un punto de vista estático, consideramos como primera alternativa incorporar dicha semántica y la información necesaria para la misma, modificando las clases y objetos que representan los elementos en el RTS, más precisamente implementando algún tipo de especialización de estos objetos y sus operaciones para cada plataforma. Esta opción, que entendemos la más adecuada para el paradigma de orientación a objetos seguido para el diseño y desarrollo del RTS 4.5, habilitaría además adoptar una definición desde el estándar basada en el concepto de especialización para el mapeo del lenguaje a las diferentes plataformas. Luego de analizar esta solución y estimar el esfuerzo necesario para realizar la tarea, descartamos esta opción para la prueba de concepto, las razones que tuvimos en cuenta fueron minimizar y simplificar el despliegue de los cambios realizados a la solución original, así como disminuir el esfuerzo de desarrollo a realizar para esta prueba. De esta forma preservamos un mecanismo sencillo y limpio para dualizar la solución con el objetivo de facilitar la adopción por terceros de los cambios propuestos, promoviendo así el uso y validación práctica de nuestra propuesta para el API Dual.

Como solución alternativa entonces, definimos ciertas estructuras de datos auxiliares e incorporamos código en los lugares específicos donde se requería la semántica para la dualización, considerando las alternativas de que una operación pudiese ejecutarse en una u otra plataforma. Utilizamos compilación condicional, de forma de minimizar los efectos colaterales de la incorporación de estos cambios al proyecto ρTTCN-3 de Go4IT, de esta forma estableciendo ciertas banderas al momento de la compilación puede construirse la solución estándar de Go4IT o la solución dualizada.

En tiempo de ejecución el comportamiento será entonces, cada vez que un elemento, que se implementa en lenguaje de plataforma, es creado por el RTS, deberá establecerse la plataforma del citado elemento basados en las estructuras de datos ya mencionadas, en función de esta información se realizará la llamada a una u otra platafoma. Por defecto un elemento que no está especificado en los archivos de configuración se gestiona como perteneciente a la plataforma nativa de la herramienta, en nuestro caso se considerará que está implementado en C/C++. Complementariamente, cada vez que una función de una interfaz sea invocada desde el RTS, se ejecutará el código con la semántica que fue agregada para la implementación del API dual, la cual tendrá la capacidad de invocar el componente en una u otra plataforma, según se debió específicar oportunamente en los archivos de configuración.

Para nuestra prueba de concepto, en el caso de la dualización de los puertos, dicha lógica fue introducida en los métodos relacionados a estos, que corresponden a las siguientes actividades `Map()`, `Unmap()`, `Send()` o `Enqueue()`, and `Receive()`.

Una de las dificultades enfrentadas en este punto fue que los nombres de las interfaces de acuerdo al estándar deben ser los mismos para ambas plataformas, y nosotros queríamos respetar estas definiciones por razones de interoperabilidad, pero si usabamos el mismo nombre para ambos, dependiendo de la implementación, podría generarse algún tipo de colisión en tiempo de enlace o compilación. La utilzación de la tecnología JNI nos ayudó a evitar este problema al diferir la llamada a la interfaz, y por lo tanto la utilización de los nombres específicos, hasta el momento de

estar ejecutando en el contexto de la plataforma Java. Para el caso de nuestra prueba de conceptos desde el código del T3RTS continuamos llamando a las funciones correspondientes a la interfaz nativa de la forma usual, mientras para a las funciones de la interfaz en la otra plataforma, Java en este caso, se utiliza una función intermedia "decorada" con el nombre de la plataforma al final del nombre de la función. Esta llamada a las funciones de las interfaces será manejada por los servicios provistos por el módulo `T3RTSDual`, el cual derivará la llamada a la función correspondiente en la JVM que maneja las ejecuciones en la plataforma Java, donde las funciones podrán utilizar el nombre definido en el estándar.

12.3. Modificaciones a otros módulos

Como explicamos al describir la clase `T3RTSDual` 12.1, las tareas de inicialización fueron introducidas en el módulo `TCICHRequired`, en particular en los métodos `TciExecuteTestCase` y `TciReset` fue necesario inicializar algunas estructuras de datos, inicializarlas aquí podría prevenirnos de no tener en cuenta algún cambio en la configuración entre dos ejecuciones de un mismo caso de prueba. En estos métodos se realizan llamadas a las correspondientes funciones de inicialización de la TRI, es necesario conocer aquí si en la implementación corriente serán utilizados elementos de ambas plataformas, a efectos de inicializar los entornos correspondientes. Esta discusión se realiza a efectos de no perjudicar el desempeño de la solución nativa al incorporar la capacidad dual, en los casos que la implementacioń no requiera del entorno de ejecución Java los objetos de JNI no serán inicializados. Para tomar esta decisión es necesario revisar de una vez la configuración de la plataforma para todos los elementos del caso de prueba o incluso de toda de la implementación.

12.4. Integración con el compilador ρTTCN-3 de Go4IT

El proyecto Go4IT está desarrollado sobre plataforma GNU en C/C++ y utiliza Autotools para automatizar la construcción de la solución [20]. Autotools es un grupo de utilidades GNU que asisten en las automatización de las tareas de construcción de binarios, donde encontramos Autoconf, Automake, Libtool y otras. Como característica destacada, además de la automatización que es el concepto de base de la herramienta, esta es muy potente en los aspectos de portabilidad.

Se modificaron scripts ya existentes de Autotools de Go4IT para crear la Solución Dual, en estos se incorporaron los nuevos archivos y se enlazaron para la construcción del binario del RTS. Se creo un script en `bash` para la invocación del configure para la generación del RTS de Go4IT con los modificadores adecuados para la compilación condicional. Además debió modificarse el Makefile para la contrucción del ejecutable para el DNSTester, al cual se debió agregar la información de enlace con las bibliotecas de JNI y los modificadores para la compilación condicional. Para revisar en forma pormenorizada las modificaciones para la construcción de los binarios referirse al Anexo 4 15.2.

Un aspecto a destacar de esta tarea es que las modificaciones se pensaron de forma de que la característica dual sea opcional, según se compile con ciertas banderas activadas o no, se construirá la solución dualizada o la original de Go4IT respectivamente. De esta forma, a pesar de disponer en el paquete de instalación del soporte para la solución, en caso de no requerirse la característica Dual, se evita la necesidad de tener instalados los paquetes de los que depende la dualización, en particular JNI y los componentes específicos en Java de la solución que se está implementando. Estos cuidados tuvieron como objetivo reducir las resistencias que pudiese generar incorporar la Solución Dual por la generación de dependencias innecesarias.

12.4.1. Cambios en los componentes de ρTTCN-3

Los cambios realizados a ρTTCN-3 se resumen en modificaciones al RTS de Go4IT, las que se encuentran empaquetadas en la biblioteca `libgo4it_p2_t3rts.a`. Un resumen de los cambios realizados para la implementación del presente prototipo, desde la perspectiva de ρTTCN-3 es presentado a continuación.

- **SA Library**
  - T3RTSPort en esta clase se inicializa la plataforma del puerto cada vez que una instancia de este es creada. Luego al momento de enviar los mensajes al SUT basado en esta información ejecuta `triMap` y `triSend` en la plataforma correspondiente.
  - TRISUTRequired esta clase es modificada para dar soporte a la ejecución desde Java vía JNI a la función `triEnqueueMsg` que encola la respuesta de los mensajes desde el SUT.
- **TM Library** en la clase `TCITMRequired`, Inicializa el ambiente JNI, desde `tciInit`, usando un hilo independiente se lanza el servicio `srv_dual`, el cual ejecuta `JenvInit` y queda a la espera por requerimientos. También limpia las referencias creadas al ejecutarse `tciFinalize`.
- **CH Library** en la clase `TCICHRequired` se carga la información de la plataforma para los puertos que forman parte de la Interfaz del sistema de Pruebas (Test Sytem Interface - TSI), y cuando corresponde inicializa las plataformas respectivas. Para esto recorre la lista general de puertos de la TSI ya disponible en el RTS y arma listas para cada plataforma.

Información pormenorizada sobre los cambios realizados en el RTS puede encontrarse en el Anexo 4 15.2.

12.5. Detalles de la Implementación relacionados con JNI

Diferentes aspectos relacionados con JNI fueron foco de nuestra atención en el proceso de desarrollo de la Solución Dual, trataremos de resumir aquí los que en nuestra opinión fueron más relevantes para este trabajo y cuya resolución nos dejó alguna practica interesante para aplicar en experiencias futuras. Considerando los aspectos básicos del uso de JNI destacan aquellos relacionados con la infraestructura, los cuales debían resolverse para dejar la solución operativa. Uno de estos aspectos, resuelto para la utilización de JNI en la implementación, fue el manejo dinámico del `classpath` y las rutas de las implementaciones en Java, es decir las referencias utilizadas en Java para acceder a los archivos de los diferentes objetos, que serán necesarios para proveer las implementaciones desde la plataforma Java por intermedio de JNI. Para disponer de esta información de forma dinámica primero consideramos la posibilidad de usar variables de ambiente con valores por defecto, que pudiesen configurarse mediante scripts, pero dadas las dificultades y cierta oscuridad que podía presentar el manejo de las mismas, especialmente su dependencia de la plataforma, optamos por disponer un archivo de configuración donde se especificase los datos necesarios para dejar operativa la implementación de la otra plataforma. El hecho de haber optado, al momento del diseño, por implementar las diferentes configuraciones de la solución en archivos, facilitó el agregado de estas especificaciones, dado que la solución ya contaba con soporte para recuperar la información por ese método.

Otro de los aspectos del manejo de la infraestructura, que ofreció cierta dificultad fue comprender el despliegue necesario para la utilización de JNI tanto al momento de desarrollo para la generación

de los binarios correspondientes, como al momento de ejecución para soportar el enlace dinámico de sus bibliotecas. Información detallada sobre este punto se encuentra en la documentación de instalación y uso de la solución 15.2. El resto de los aspectos que consideramos básicos respecto al uso de JNI fueron absorbidos por la utilización de un conjunto de bibliotecas ya existentes, como veremos en detalle en la Sección 12.5.1.

Dentro de los aspectos avanzados de JNI revisaremos la gestión de los diferentes ambientes de ejecución y su interoperabilidad, en especial nos detendremos en la gestión del entorno de ejecución para múltiples hilos (multithreading), así como en la invocación, desde la máquina virtual de Java creada mediante JNI (Java Virtual Machine - JVM), a la misma instancia de código que inicia la ejecución en C/C++, mediante el uso de punteros a función para preservar el contexto de ejecución en la aplicación invocadora.

Veremos a continuación como fué nuestra aproximación inicial a la tencnología JNI, las diferentes bibliotecas utilizadas e implementadas, para luego detenernos en los aspectos avanzados del uso de JNI.

12.5.1. Bibliotecas que encapsulan el uso de JNI

.

Nuestro punto de partida para el uso de JNI no fue directamente su API, sino un conjunto de bibliotecas construidas sobre esta durante el proyecto de grado **Generación Automática de CODECS para TTCN-3 / Java** [26].

Como desarrollamos en el punto 8.1 este proyecto resolvió un caso concreto de interoperabilidad de plataformas para TTCN-3, entre el T3DevKit y el compilador Java TTWorkBench de Testingtech [39]. Participamos en el proyecto durante la fase de definición y diseño de la solución, aportando nuestra experiencia en el lenguaje, su composición y la localización de los puntos de interoperabilidad, también propusimos algunas ideas para el diseño de la solución. Esta participación fue posible ya que el mencionado proyecto se desarrolló entre nuestra participación en los proyectos de construcción del compilador ρTTCN-3 y la Solución Dual, de esta forma al plantearse este proyecto habíamos acumulado cierto conocimiento sobre TTCN-3 y en particular de la implementación realizada en Go4IT, que nos permitió realizar los aportes mencionados.

Con la orientación de nuestros tutores capitalizamos esta actividad para que el resultado de dicho proyecto fuese usado posteriormente en nuestra implementación de la Solución Dual, como base para la utilización de JNI. El proyecto requería el desarrollo de un conjunto de funciones para facilitar el uso de JNI que fueron agrupadas en una biblioteca llamada `JNIUtils`, su encapsulamiento facilitó el uso de las mismas en este proyecto y también su re-uso en nuestro trabajo sobre la plataforma dual que se desarrollase a continuación. Utilizando esta biblioteca de base, se implementaron otras cuyo cometido inicial era brindar las funcionalidades necesarias para la utilización de JNI en la construcción de un *wrapper* estático para algunas de las interfaces de TTCN-3, además de estas necesidades al definirla se tuvieron en cuenta algunos aspectos que requeriríamos para el desarrollo del *wrapper* dinámico.

Más allá de estas previsiones, en el proceso de desarrollo del *wrapper* dinámico debímos resolver algunos aspectos que quedaban claramente fuera del alcance de este proyecto, para lo cual contamos con el apoyo de los integrantes del equipo de este que acumulaban una experiencia muy valiosa en el uso de JNI. Algunos de estos aspectos son la implementación de la interoperabilidad con las interfaces **TRI/SA**, **TCI/CH** y **TCI/TM** de ρTTCN-3 en la plataforma Java, en el otro sentido la implementación de la interoperabilidad con el compilador C/C++ para las operaciones de respuestas como `triEnqueueMsg` y `triTimeout` también de la **TCI/CH**, ambos sentidos necesarios para la implementación del prototipo dual.

A continuación enumeramos las bibliotecas que dan soporte al uso de JNI y sus responsabilidades.

JNIUtils Esta biblioteca implementa las funciones básicas necesarias para la uitilización de JNI de forma más sencilla desde las aplicaciones C/C++. En esta biblioteca se crean, recuperan y destruyen los entornos de ejecución Java necesarios para instanciar y ejecutar implementaciones en Java desde C/C++. Aquí también se automatizan conversiones de datos no escalares, conversiones de cadenas y llamadas a funciones con diferentes tipos de retorno, facilitando la invocación de estas funcionalidades desde las bibliotecas de más alto nivel o desde las aplicaciones. Por último esta biblioteca también cuenta con una primitiva para el registro dinámico de funciones nativas, lo cual permite la utilización de funciones disponibles en bibliotecas dinámicas, como las .so de GNU/Linux.

Dual_JNI_C_Java Esta biblioteca implementa la llamada desde el ambiente C/C++ de las funciones de las interfaces de TTCN-3 correspondientes, que se encuentran implementadas en Java. Por ejemplo aquí encontraremos la implementación de las llamadas para la **TRI/SA** y **TRI/PA** requeridas (Required).

Dual_JNI_Java_C Esta biblioteca publica la llamada de las funciones nativas C/C++ de la interfaz TTCN-3 correspondientes en JNI, de forma que estas puedan ser invocadas desde Java. Para posibilitar esto implementa las conversiones necesarias de tipos de datos TTCN-3 y la llamada a las funciónes nativas. Aquí encontraremos la implementación de las llamadas para **TRI/SA** y **TRI/PA** provistas (Provided), donde podemos destacar la función `triEnqueueMsg` que envía las respuestas desde el SUT al TE, es en relación con esta función que veremos, en la Sección 12.5.2, el problema y la solución de la ejecución de la función en la instancia ya existente del proceso nativo.

Haber contado con estas bibliotecas como base para nuestro trabajo, significó una reducción muy importante en el esfuerzo total de implementación y estudio de la tecnología JNI para la construcción de la Solución Dual. Para obtener más información sobre estas bibliotecas puede consultarse la documentación del mencionado proyecto [26], en su capítulo **5**.

12.5.2. Interoperabilidad de los ambientes de ejecución

En este punto describiremos más en detalle como se resuelven los aspectos de interoperabildad entre las plataformas usando JNI, en particular nos dentendremos a analizar como se logra desde cada ámbito de ejecución el acceso a la otra plataforma, la información presentada aquí refleja nuestra experiencia concreta y en general se apoya en la documentación de JNI [36]. Un aspecto común a la interoperabilidad desde ambos ambientes es la conversión de datos entre las plataformas, esta se resuelve de forma encapsulada mediante el uso de la biblioteca `JNIUtils`. Esta biblioteca extendiendo las características de JNI, permite exponer y actualizar los datos entre las plataformas, mediante un conjunto de primitivas que implementan las operaciones básicas para la recuperación y actualización de tipos estructurados y objetos, convirtiendo o asistiendo en las conversiones necesarias a estos efectos. Por otro lado los módulos `util` de las bibliotecas correspondientes a la implementación Dual, `Dual_JNI_C_Java` y `Dual_JNI_Java_C`, resuelven las conversiones específicas para los tipos de TTCN-3 que se necesitan intercambiar en cada caso, utilizados por ρTTCN-3 para acceder a las plataformas de base como BinaryString y otros.

Invocaciones desde C/C++ a Java

Para nuestra prueba de conceptos, en la mayor parte de los casos, estaremos situados del lado de la plataforma nativa C/C++ y precisaremos invocar diferentes funcionalidades implementadas en la plataforma Java que conforma la interfaz provista (Provided) para la plataforma Java. Para este caso la biblioteca `JNIUtils` desarrollada usando JNI provee un ambiente de ejecución Java, además en varios casos este entorno deberá conservar el estado de la ejecución actual entre diferentes llamadas, lo cual también es soportado por esta biblioteca. Para esto las funciones de la biblioteca `Dual_JNI_C_Java` trataran siempre de utilizar el entorno de la JVM que fue definido al inicializar el módulo `T3RTSDual`, solo en caso de que esta no esté aún disponible será inicializado y utilizado uno nuevo.

En este entorno de ejecución, provisto por las bibliotecas, serán inicializados los objetos Java que implementan las interfaces, cada vez que se inicialice el sistema de pruebas. A partir de ese momento cada vez que se quiera ejecutar una función de la interfaz en la plataforma Java se realizará una llamada a su función correspondiente en la biblioteca `Dual_JNI_C_Java` la cual, además de los parámetros de la función, deberá recibir referencias al ambiente JNI (JNIEnv) y al objeto que implementa la interfaz que se han inicializado previamente. Finalmente estas funciones hacen uso de las primitivas de `JNIUtils` para realizar las conversiones de datos e invocaciones necesarias en la plataforma Java.

Invocaciones desde Java a C/C++

En el caso de las invocaciones desde Java a C/C++ se diferencia claramente una situación básica, cuando una aplicación Java independiente usa funciones nativas de forma directa, de otra más compleja donde la llamada desde Java a C/C++ se realiza desde una aplicación, ejecutándose en una máquina virtual, que a su vez está embebida en una aplicación nativa mediante el uso de JNI, trataremos ambas situaciones por separado a continuación.

Para el caso más simple podemos considerar como ejemplo una aplicación Java que realiza la invocación a una función nativa para resolver alguna funcionalidad que es muy costosa o no puede resolverse directamente en Java. Esta invocación se realiza mediante la interfaz para la ejecución de metodos nativos de JNI (Native Methods Interface), la cual permite tener acceso desde Java a los recursos de entorno operativo donde se ejecuta la máquina virtual y usar los servicios de este para ejecutar código binario compilado para el, contando así con acceso a los servicios nativos provistos por el entorno operativo. En el caso de TTCN-3 un ejemplo de esto puede darse en una herramienta TTCN-3 Java, como TTWorkBench de Testingtech [39], que requiriese el uso de funciones externas en lenguajes de bajo nivel como C/C++ para la implementación de protocolos de red.

En el segundo caso se trata de una aplicación nativa, como puede ser un navegadior de internet, que requiere ejecutar una aplicación en Java, para lo cual ejecuta una máquina virtual embebida mediante el uso de la interfaz de invocación de JNI (Invocation Interface). A su vez esta aplicación Java puede requerir algún servicio provisto por el sistema operativo donde se ejecuta el navegador, para lo cual deberá recurrir a la ejecución de código nativo nuevamente mediante la interfaz para la ejecución de métodos nativos de JNI. En el caso de TTCN-3 podemos encontrar esta situación en un compilador C/C++, que requiera usar una aplicación en Java para la prueba de Web Services, la cual deberá luego ejecutar las interfaces requeridas del estándar que son provistas por el compilador desde la aplicación C/C++.

Para que una función nativa este disponible en Java a través de JNI, deberá generarse un *wrapper* para la misma, el cual debe incluir ciertos parámetros complementarios que brindan al *wrapper* el acceso a el ambiente Java y a los objetos desde donde es invocado, luego deberá a su vez

compilarse con los modificadores correspondientes y por último la función *wrapper* debe exportarse dentro de una biblioteca. Los headers necesarios para este *wrapper* podrán crearse usando la utilidad `javah` [36] con el modificador -jni o con la asistencia de algún IDE específica.

Para que las funciones así definidas puedan ejecutarse desde Java la biblioteca que las contiene debe ser cargada en tiempo de ejecución mediante la instrucción `System.loadLibrary`. Luego aún falta localizar estas funciones en el entorno Java, para lo cual existen algunas opciones que veremos a continuación. El entorno Java puede localizar las funciones automáticamente buscando por su nombre entre las bibliotecas cargadas, de esta forma las funciones de la biblioteca cargada podrán ser ejecutadas directamente desde Java, esta opción si bien evita cargar bibliotecas de forma innecesaria, puede no ser conveniente en algunas situaciones donde queremos evitar el castigo en el desempeño al momento de ejecutar la función. Otro método de localización explicito consiste en utilizar la función *RegisterNatives()*, la cual permite además localizar dinámicamente una función de las bibliotecas cargadas con la instrucción `System.loadLibrary` y también una función que resida en la función invocadora, para el caso de una aplicación nativa que hospeda el ambiente Java que está realizando la invocación a la función nativa, En este caso las funciones no precisan ser exportadas ni seguir la convención de nombres de JNI, pero si requieren seguir el protocolo de los parámetros adicionales del *wrapper* y tener el modificador *JNICALL*.

Para algunos casos es recomendado utilizar la segunda opción [36]

- cuando una aplicación nativa tiene una máquina virtual incrustada, esta puede no encontrar las funciones que residan en la aplicación, pues solo busca en las bibliotecas nativas cargadas.
- para actualizar dinámicamente, en tiempo de ejecución, la función a la que llama el método nativo.
- para controlar el momento en que se realiza el enlace, evitando que este se realice al momento de ejecuión.

En nuestro caso aplica la primera opción, las funciones de la interfaz requerida del estándar residen en el compilador ρTTCN-3 nativo, que serán invocadas desde la máquina virtual incrustada en el mismo para ejecutar el mapeo de las interfaces provistas en Java, por este motivo debimos incorporar en estos casos la utilizacón de *RegisterNatives()* como mecanismo para la invocación de las mismas.

Nos queda aún otra situación donde ninguno de estos métodos se mostró efectivo, es cuando se necesita ejecutar las funciones dentro de una instancia de un proceso previamente existente, donde residen variables de tipos complejos con datos, instancias de objetos u otro tipo de recursos que es requerido reusar en la ejecución de la función nativa, será necesario recuperarlos utilizando técnicas más complejas. Si bien para estos casos JNI propone la opción de usar *RegisterNatives()*, esta no se mostró efectiva usada de forma directa en los casos donde se trata de recuperar una instancia ya existente y utilizar datos complejos.

En el caso del RTS nativo C/C++ y la utilización de implementaciones en la plataforma Java, por definición deben proveerse los objetos Java de forma dinámica, ya que queremos brindar la capacidad de cambiar la plataforma de estos elementos sin necesidad de compilar nuevamente la solución. Al ser el RTS nativo C/C++ una biblioteca dinámica y requerirse además la utilización de los tipos de datos de los identificadores, la solución básica ofrecida por JNI no es aplicable en este caso y se debió buscar otra alternativa. La solución encontrada fue la utilización combinada de *RegisterNatives()* y punteros a funciones en C/C++, estos son usados para llamar desde la plataforma Java a funciones que se ejecuten en instancias ya existentes de procesos en la plataforma nativa desde la que fue creada la máquina virtual embebida. Para esto debá recuperase estos

punteros a función de forma dinámica, la forma que se encontró para resolver esto, fue utilizar la técnica del *RegisterNatives()* con la función `GetAddress()` en la aplicación nativa, que recupera la dirección en memoria de la función a ejecutar. Cuando se requiere llamar una función C/C++ desde Java, como retorno de una llamada anterior (callback), en la llamada inicial desde C/C++ se obtiene, con el operador de desreferenciación, la dirección de la misma y se provee al entorno Java mediante el uso de la función antes mencionada.

12.5.3. Gestión de múltiples hilos

Durante la implementación debimos resolver diferentes aspectos relacionados con la visibilidad y disponibilidad del entorno Java, la máquina virtual embebida provista por JNI, desde los diferentes componentes del RTS de ρTTCN-3. Luego de una implementación inicial sencilla y directa donde asumimos que el ambiente creado con JNI podría usarse desde toda la solución, se generaron algunos problemas que nos obligaron a replantearnos el camino que habíamos utilizado. Nuestros primeros prototipos se habían orientado a localizar las referencias al entorno JVM en alguna clase de la raiz de la jerarquía del RTS, para que de esta forma estuviese disponible para todas las clases del mismo. Para esto buscamos una clase con estas características para agregar dicha lógica, dada la falta de documentación y nuestro desconocimiento sobre el diseño del RTS, exploramos la solución utilizando el depurador *GDB* [21] para seguir el curso de la ejecución y determinar el lugar exacto donde introducir los cambios. De esta forma determinamos los puntos donde el compilador inicializaba los elementos candidatos a ser implementados en las plataformas, también localizamos la invocación a las interfaces, logrando así familiarizarnos con el diseño del compilador y los detalles de la programación con un mínimo costo para las condiciones que teníamos.

Una vez que el sistema quedó operativo, al ejecutar las primeras pruebas verificamos que en ciertas condiciones las referencias a los objetos de JNI quedaban obsoletas, luego de algún relevamiento sobre este tema, analizando problemas similares descritos en algunos foros de Internet, llegamos a la conclusión de que estábamos cometiendo un error en la utilización de la infraestructura de JNI relacionado con la naturaleza multi hilo de TTCN-3.

Con esta idea en mente pudimos verificar en la documentación de JNI [36], en el capítulo **7 The Invocation Interface** punto **7.3 Attaching Native Threads** y el capítulo **8 Additional JNI Features** punto **8.1.4 Obtaining a JNIEnv Pointer in Arbitrary Contexts**, que las referencias a la JVM deben almacenarse en variables globales y usarse siempre desde el mismo hilo de ejecución, allí también pudimos evaluar algunos mecanismos para resolver esta limitación.

El problema encontrado fue que la utilización directa de JNI no puede compartirse desde diferentes hilos, debido a su arquitectura al intentar utilizar un ambiente JVM creado en un hilo desde otro diferente se produce una excepción de tiempo de ejecución dentro de este.

La recomendación realizada por la documentación de JNI [36] es que se adjunten los hilos de ejecución a la JVM correspondiente mediante el uso del método `AttachCurrentThread`, también aquí se realizan recomendaciones de valor para la recuperación del entorno JNI.

Con este diagnóstico nos dedicamos a analizar soluciones al problema planteado, teniendo además en consideración nuestras posibilidades y recursos disponibles.

En primera instancia analizamos la recomendación de JNI, si bien la misma era clara y no parecía tener contraindicaciones de ningún tipo, la biblioteca `JNIUtils` que usamos para tener soporte JNI no consideraba la posibilidad de realizar la operación de adjuntar un hilo a una JVM (`AttachCurrentThread`), inicialmente no se encontraba en el alcance de nuestros conocimientos modificar estas bibliotecas para dotarlas de dicho soporte. Las alternativas que teníamos para modificar dicha biblioteca eran recurrir a recursos fuera del proyecto o realizar una inversión de tiempo considerable para estudiar la tecnología y modificarla.

Nos planteamos como una solución alternativa mantener el uso de la JVM en un único hilo de ejecución, si bien esta alternativa podría solucionar el problema, debía lidiar con la naturaleza asíncrona de los comportamientos esperados en ρTTCN-3, para considerar este aspecto decidimos implementar un servicio que serializase el acceso a la infraestructura JNI mediante la utilización de semáforos, manteniendo de esta forma el comportamiento asincrónico del lado de ρTTCN-3. Se implementó entonces un servicio para el acceso a JNI que mantiene en un único hilo las invocaciones a JNI, se utilizarón las bibliotecas de pthreads de GNU/linux, *POSIX threads* [8], para la gestión de los hilos y los semáforos correspondientes.

De esta forma nuestra solución final resolvió el uso de JNI al declarar e inicializar el entorno de Java provisto por JNI en la clase `T3RTSDual`, referenciándolo desde variables con visibilidad global y ofreciendo el uso de los mismos como un servicio para los otros hilos de ejecución Si bien la solución implementada puede tener algunas limitaciones para una implementación real, especialmente en su robustez y capacidad de concurrencia real, la entendimos suficiente para nuestra prueba de conceptos. Queda pendiente entonces la modificaciń de las bibliotecas que dan soporte a JNI, en particular de **JNIUtils**, de forma que estas acepten requerimientos desde diferentes hilos aprovechando las capacidades recomendadas por JNI para esto.

12.5.4. Implementación de Funciones

Para esta tarea la biblioteca JNI implementa la correspondencia entre objetos y tipos de datos, esta es una implementación genérica que basada en los nombres del paquete y las interfaces crea dinámicamente los objetos necesarios en cada caso. En un sentido para invocar desde funciones C interfaces implementadas con métodos en Java, el módulo llamador debe conocer las clases que implementan las interfaces. En el otro sentido para invocar desde Java funciones de las interfaces implementadas con métodos en C/C++, el módulo llamador en Java necesita cargar la biblioteca dinámica donde se implementan las interfaces. Esta información se encuentra almacenda en los archivos de configuración del RTS.

12.6. Implementación en Java

La biblioteca TTCN3_TRI_Java implementa las funcionalidades de nuestra prueba de conceptos en la plataforma Java, la misma incluye la implementación del mapeo en Java de las interfaces del estándar y el socket para el puerto de nuestro experimento que describiremos en 13.

- **TRI**
  - textbfData, implementa las interfaces en Java para intercambiar cada uno de los tipos de datos de ρTTCN-3 con objetos en Java.
  - textbfPA, implementa las operaciones necesarias para comunicar el ETS con la Plataforma en Java.
  - textbfSA, implementa las operaciones necesarias para la comunicación entre el ETS y el SUT Adapter para Java.
  - textbfUtils, contiene algunas funciones útiles para la implementación de las interfaces.
- **Implementación de adaptador**
  - textbfDNSTester, basados en la implementación del DNSTester para Go4IT en C/C+ se implmenta la conexión e interacción con un puerto socket de acuerdo a lo descrito para el ejeplo del DNSTester.

Capítulo 13

Experimento con el prototipo ρTTCN-3 Dual Implementación del DNSTester

El caso de prueba del DNSTester es utilizado como ejemplo en el libro de "An Introduction to TTCN-3" [41], el mismo es muy popular en la comunidad TTCN y cumple con los requerimientos necesarios para experimentar con el prototipo de la Solución Dual, además está en el ámbito del testing de redes que nuestro centro de atención y fue implementado en el proyecto Go4IT como caso de prueba para el compilador ρTTCN-3.

El DNSTester es un sistema para la prueba de un servicio DNS de una red IP, el mismo consta principalmente de una Adaptador al SUT compuesto por un cliente socket, que se conectará al Servidor DNS y realizará consultas al mismo, esperando y validando su respuesta, de acuerdo a al comportamiento esperado del servicio, para las condiciones de la prueba. En la Figura 13.1 podemos observar un diagrama del SUT.

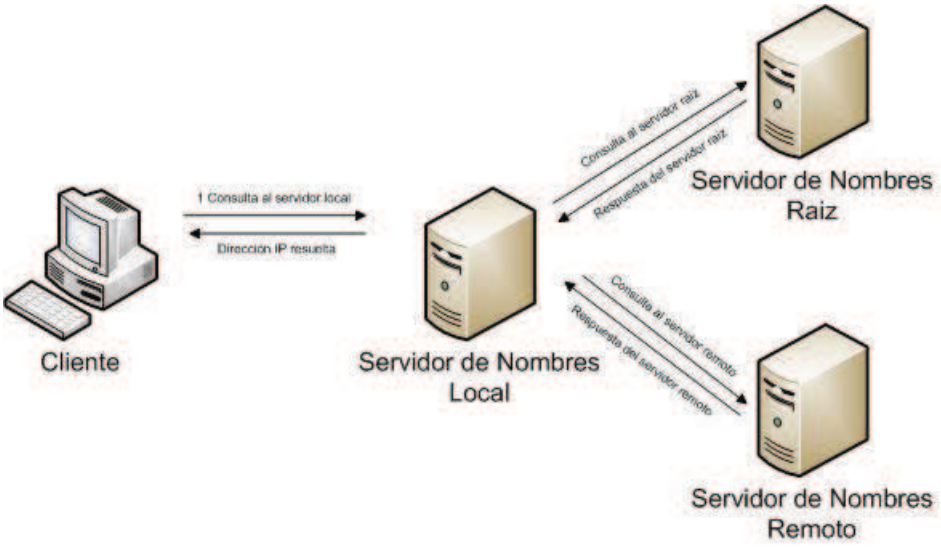

Figura 13.1: Caso de prueba DNSTester

El siguiente diagrama MSC (Message Secuence Chart) nos permite deducir que el test solo involucra el Tester y el Servicio Local de Nombres, además podemos ver que el Tester envía una consulta sobre www.nokia.com al Servicio Local de Nombres y a continuación recibe como respuesta

la IP correcta.

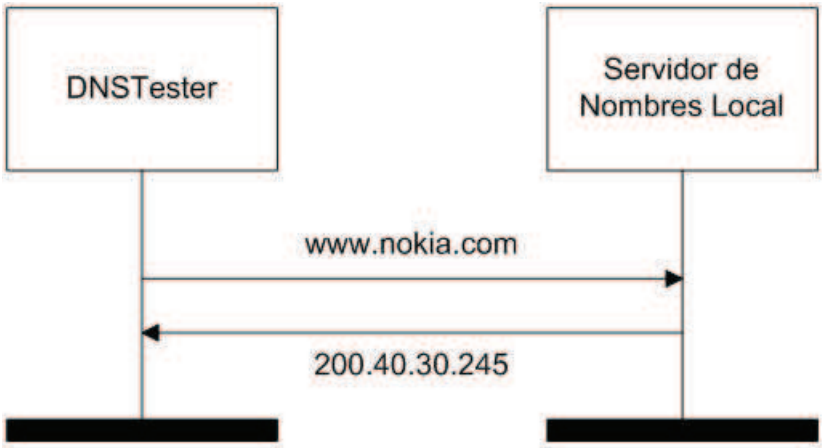

Figura 13.2: Diagrama MSC para el DNSTester

Seleccionamos entonces el caso de prueba del DNSTester para realizar un experimento que validara el prototipo del compilador dual, el mismo nos permitía realizar las pruebas necesarias, alternando la plataforma para la implementación del Adaptador para el puerto del DNSTester.

Para la realización de esta prueba nos propusimos mantener el código TTCN-3 utilizado en Go4IT para las pruebas de ρTTCN-3 sin cambios, garantizando de esta forma la portabilidad. Implementar el Adaptador para el cliente socket en Java y realizar las configuraciones necesarias para que el compilador Dual utilizara alternativamente las implementaciones en C/C++ y en Java, esperando que los resultados fuesen los mismos para los casos en que la prueba fuese exitosa o fallara, de acuerdo a las condiciones en las que se realizara la prueba.

Para poder verificar las pruebas correspondientes en cada plataforma, agregamos trazas en la implementación que nos permitiesen distinguir cuando estábamos ejecutando el código en cada plataforma y los resultados que se van obteniendo en cada caso.

El experimento se comportó de la manera esperada, cuando la configuración para el puerto del DNSTester no está presente, el caso de prueba se comporta de la manera usual ejecutando el código en C/C++ del CDGen, cuando se agrega la configuración para que el que el puerto DNSTester utilice la implementación en Java del cliente socket esta es ejecutada mediante la invocación de las interfaces estándares en Java. Los resultados del caso de prueba también fueron consistentes, resultando exitosas o fallidas las ejecuciones para ambas plataformas de acuerdo a las condiciones de la prueba en cada caso.

La Figura 13.3 *Experimento - DNSTestr DUAL* muestra el diagrama de componentes de la solución para ambas plataformas. Los componentes de color verde corresponden a la implementación Go4IT del caso de prueba, estos son las bibliotecas del RTS en C/C++, el T3DevKit y el CDGen donde se realiza la implementación de los Adaptadores en C/C+. Los componentes en naranja corresponden a la implementación del caso de prueba en Java, los cuales incluyen la biblioteca del RTS para Java, y la implmentación *ad hoc* en Java de los Adaptadores. Por último, en gris pueden verse las configuraciones para el RTS Dual, esto incluye la localización de la implementación del las bibliotecas den RTS en Java, la localización de la implementación de los Adaptadores en Java y la configuración de la plataforma para el puerto del DNSTester que determina la plataforma en la que este se ejecutará en cada caso. El resto de los elementos que se implementan en plataformas

de base, como los CoDecs, al no estar configurados explícitamente en este archivo se tomarán por defecto de la plataforma de base del compilador, en el caso de ρTTCN-3 esta es C/C+ y los CoDecs están resueltos automáticamente usando el T3DevKit y el CDGen de Go4IT.

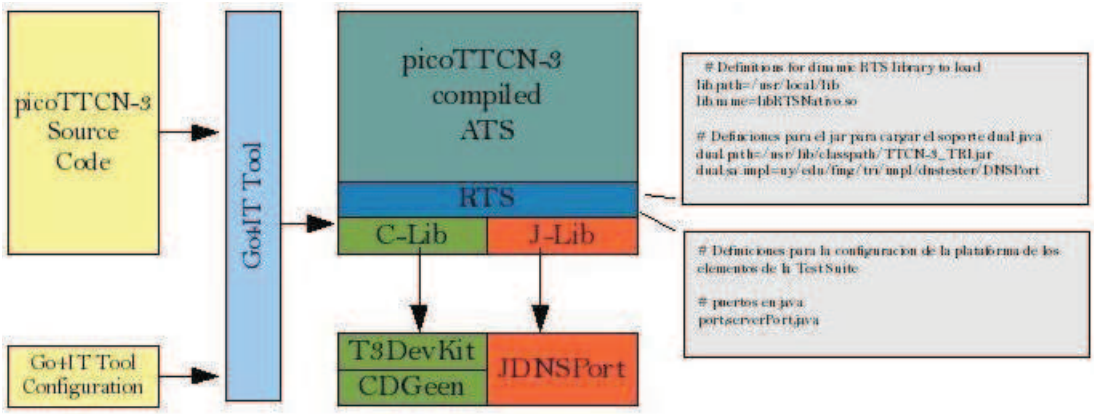

Figura 13.3: Experimento - DNSTestr DUAL

13.1. Código fuente y archivos de configuración

A continuación se presentan algunos detalles relevantes de la implementación, primero se muestra la implementación del Adaptador para el DNSTester en Java, luego la implementación del mismo en C/C++ y por último los archivos de configuración junto a la traza generada en cada caso.

13.2. DNSTester Source Code Map - Java

```
public TriStatus triMap(TriPortId compPortId, TriPortId tsiPortId) {
System.out.println("***** triMap *****");
        System.out.println("    CompPort: " + compPortId);
        System.out.println("    TsiPort: " + tsiPortId);
                this.compPortId = compPortId;
                this.tsiPortId = tsiPortId;
                this.nameserver = "200.40.30.245";
                DNSServerAddress =
InetAddress.getByName(nameserver);
                if (connected) {
                    return new TriStatusImpl("Already Connected
Port");
s// automatically set the Error Code
                }
                this.triCommunicationTE = new
TriCommunicationTEImpl();
                this.start();
        return new TriStatusImpl(TriStatus.TRI_OK);
```

```
}
public TriStatus triRaise(TriComponentId componentId, TriPortId
tsiPortId,
TriAddress sutAddress, TriSignatureId signatureId,
TriException exception) {
System.out.println("***** triRaise *****");
return new TriStatusImpl(TriStatus.TRI_ERROR);
}
```

13.3. DNSTester Source Code Map - C

```
/** Map the DNSPort to a TTCN-3 port.
 *
 *  This function is called when the Test Executable requests to map
the port
 *  to a TTCN-3 port.
 *
 *  This function:
 *   - create a UDP socket connected to the DNS server
 *   - start a listening thread for monitoring messages received on
this socket
 *
 *  @note This port can mapped to only 1 TTCN-3 component at once
(this
 *   is a design constraint)
 *
 *  @return true if successful
 *
 *  @see dns_thread(), Unmap()
 */
bool DNSPort::Map (const PortId& port_id)
{
// Allow only one connection
if (connectedPort_ != NULL) {
perror ("DNSPort: cannot map to more than one component");
return false;
}

// Create and connect a UDP socket
socket_ = socket (PF_INET, SOCK_DGRAM, 0);
if (socket_ == -1) {
perror ("DNSPort: Error: unable to create socket");
return false;
}

struct sockaddr_in dns_server_addr;
if (get_nameserver_address (dns_server_addr) == false) {
```

```
return false;
}

if (connect (socket_, (struct sockaddr*) &dns_server_addr,
sizeof (dns_server_addr)) != 0) {
cerr << "DNSPort: connect() failed" << endl;
close (socket_);
socket_ = -1;
return false;
}

// Start listening thread
if (pthread_create (&thread_, NULL, &dns_thread, this)) {
cerr << "DNSPort: pthread_create() failed" << endl;
close (socket_);
socket_ = -1;
return false;
}

// Remember the port id
connectedPort_ = &port_id;

return true;
}
```

13.4. Configuración de las bibliotecas del RTS y las implementaciones en Java

```
file configRTS.properties

# picoTTCN-3 Run Time System Configuration File

# Definitions for dinamic RTS library to load
lib.path=/usr/local/lib
lib.name=libRTSNativo.so

# Definiciones para el jar para cargar el soporte dual java
dual.path=/usr/lib/classpath/TTCN-3_TRI.jar
dual.sa.impl=uy/edu/fing/tri/impl/dnstester/DNSPort
```

13.5. Configuración de la plataforma para los elementos del lenguaje

```
file configDUAL.properties

# Definiciones para la configuracion de la plataforma de los
elementos de la Test Suite
```

```
# puertos en java
port,serverPort,java
```

Capítulo 14

Evaluación de la Solución Dual

En este capítulo revizaremos las leccciones aprendidas al definir y construir la Solución Dual, sus trabajos futuros, finalmente realizaremos una evaluación de la misma presentando sus principales virtudes y defectos.

14.1. Lecciones Aprendidas

Durante las primeras etapas del trabajo pudimos verificar en la práctica, como nos habían trasmitido inicialmente desde la comunidad TTCN-3 y la industria en base a su experiencia, que el uso entre diferentes plataformas de adaptadores, bibliotecas o herramientas era demasiado costoso en términos del esfuerzo de la construcción de `wrappers` a medida para cada implementación, lo cual no estimula el re-uso de soluciones entre distintas plataformas. Esta primera constatación validó tempranamente la necesidad de ofrecer una solución como la propuesta para brindar soporte multiplataforma de un forma sencilla y especialmente poco costosa.

14.1.1. Evaluación del diseño elegido y sus alternativas

La opción tomada de localizar la Solución Dual en el sistema de tiempo de ejecución, en contraste con ubicarla en la etapa de tradución como habíamos evaluado inicialmente, brindó a la solución mayor dinamismo y flexibilidad. Esta opicón también significó que la misma fuese más pequeña y localizada en términos de código, siendo este un factor importante para la potencial implementación de la misma en otros compiladores y herramientas. Otro factor que se ve reforzado con esta alternativa es la transparencia de la solución para los equipos dedicados al diseño y desarrollo de los casos de prueba, segregando claramente la especificación de la plataforma de las implementaciones del código TTCN-3 y facilitando en lo metodológico la separación de roles entre los especialistas en la realización de pruebas y los especialistas en las plataformas.

14.2. Trabajos Pendientes y Futuros

Considerando el interés despertado por el compilador ρTTCN-3 y la solución Dual en la comunidad TTCN-3, nos parece interesante plantear brevemente los trabajos pendientes para completar la Dualización de la plataforma y algunas perspectivas para su uso en trabajos futuros.

14.2.1. Completar la dualizacion para ρTTCN-3

Si bien desde el punto de vista de generar un prototipo para validar experimentalmente la propuesta de la Solución Dual el trabajo realizado fue suficiente, desde la perspectiva del compilador ρTTCN-3 aún quedan áreas por cubrir tanto en lo funcional como en lo técnico.

Alcance dualizacion para ρTTCN-3

Algunas tareas quedan pendientes para completar la interoperabilidad en forma nativa en ρTTCN-3. Para esto es necesario que las APIs TRI y TCI sean completamente duales, esto implica modificar cada una de las bibliotecas que utilizan las interfaces de acuerdo al siguiente cuadro:

- SA Library (terminado)
- PA Library (realizado parcialmente)
- TM Library (pendiente)
- CD Library (realizado parcialmente)

Orientación a objetos

El escaso tiempo y la brecha de conocimientos del diseño de algunos módulos de ρTTCN-3, especialmente en el RTS, junto con nuestra inexperiencia en el paradigma de objetos, nos llevó en algunas oportunidades a optar por realizar la implementación del API Dual bajo un paradigma imperativo o al menos no tan puramente orientado a objetos como entendíamos que era la mejor solución, por estos motivos queda pendiente el rediseño y la reimplementación de algunas partes de la solución bajo un paradigma de objetos más puro.

La implementación del módulo dual, si bien es una clase C++, está solo parcialmente diseñada bajo el paradigma de objetos, falta en varias partes ser más exigente con la aplicación del mismo.

Durante la evaluación de la forma de dualizar las interfaces del RTS debímos descartar el re-diseño de los objetos del mismo, entendemos que esta es la solución más adecuada desde la perspectiva de ofrecer una solución definitiva al problema.

Implementar soporte múltiples hilos con JNI

De acuerdo a lo descripto en el punto 12.5.3 *Gestión de múltiples hilos*, la naturaleza de TTCN-3 requiere la getión multihilos del RTS, en especial en la interacción con las interfaces. Este punto fué solucionado en primera instancia con una técnica de la plataforma de base del compilador, por consideraciones de tiempo en la producción de la prueba de conceptos, quedando pendiente para una etapa posterior su implementación utilizando las características brindadas por JNI para el manejo de múltiples hilos, esta solución será más robusta y mejorará la mantenibilidad de la solución.

14.2.2. Trabajos futuros - Solución Dual

El concepto de fondo de la Solución Dual es brindar acceso simultaneo a implementaciones en las dos plataformas definidas por el estándar, esta idea y la solución propuesta se pensaron para su re-uso en otros compiladores y herramientas, de forma de brindar en forma nativa a estas portabilidad entre plataformas. Esto significaría para los compiladores Java habilitar el uso de implementaciones y bibliotecas en código C/C++, proveyendo implementaciones de la interfaz dual, en el otro sentido

para los compiladores C/C++ habilitar el uso de implementaciones y bibliotecas en código Java mediante el uso de interfaces duales, mientras desde la perspectiva de las herramientas de gestión de pruebas permitiría tener la capacidad de manejar compiladores en la otra plataforma.

La misma idea utilizada para proveer interoperabilidad a las herramientas de las plataformas definidas en el estándar, podría utilizarse para extender las implementaciones a otros lenguajes de plataforma sin necesidad de implementar una herramienta completamente en estas ni modificar el estándar, por ejemplo podrían utilizarse implementaciones en C# mediante la extensión del RTS y sus respectivas bibliotecas para este lenguaje.

14.3. Evaluación de la solución construida

Luego de la construcción del prototipo del API Dual y los experimentos realizados con el ejemplo del DNSTester, podemos afirmar que la nueva interpretación propuesta para el estándar efectivamente habilita el re-uso de bibliotecas y herramientas entre las diferentes plataformas definidas en el mismo, con un esfuerzo razonable en la implementación. Entendemos que esta solución para la portabilidad de implementaciones entre las plataformas de base, es suficientemente transparente desde la perspectiva del usuario y el desarrollador TTCN-3; ya que la misma, mediante una sencilla configuración, permite separar las tareas de los equipos responsables de la creación de los casos de prueba abstractos en TTCN-3 de aquellos responsables de la creación de los adaptadores en la plataforma de base específica.

Desde el punto de vista de los proveedores de herramientas TTCN-3, aquellos que incorporen esta solución a sus plataformas, podrán reusar herramientas y bibliotecas de otras plataformas de base de forma sencilla y a un bajo costo. Desde una perspectiva más general, si estos conceptos son implementados ampliamente, la comunidad de usuarios de TTCN-3 estará en condiciones de aprovechar los mejores servicios y soluciones ofrecidos por cada plataforma, en lugar de tener que elegir previamente la plataforma de base para todas las partes de un proyecto.

14.3.1. Repercusiones del trabajo

En el ámbito académico, presentamos la Solución Dual en la Conferencias de Usuarios de TTCN-3 del año 2008: **TTCN-3 UC 2008: "TTCN-3 Tools Interoperability Between Java and C/C++ Platforms"** [32], la presentación despertó interés entre los participantes y generó un debate en torno al factibiliad de su solución de acuerdo a nuestra propuesta, lo cual mostró que el tema era percibido como una necesidad concreta en la comunidad TTCN-3.

Este debate se centró en entender y validar como se resuelve dinámicamente el acceso a las plataformas y en consecuencia que tan automática resulta la invocación de componentes en las mismas con la solcuión propuesta, este aspecto determina en definitiva que tan atractiva es la solución para los distintos actores.

Se pusieron en consideración otros temas propuestos en la presentación, como la posibilidad de utilizar los conceptos planteados para birndar capacidad Dual a otras herramientas del mercado que requerían de la misma. En este sentido se nos llegó a consultar si teníamos una propuesta concreta, ya que algunas perspectivas del mercado de telefonía móvil, relacionadas a la liberación de la tecnología 3G, hacían interesante resolver este tema en lo inmediato para algunas empresas. Una vez finalizada la discusión quedaron varios interlocutores interesados en los resultados a los cuales no pudimos darle seguimiento individual, como alternativa mantuvimos a la comunidad TTCN-3 al tanto de nuestros avances en el desarrollo del prototipo a través del grupo de correos de ETSI para el lenguaje (*TTCN3@LIST.ETSI.ORG*), donde actualizamos los mismos hasta le finalización y liberación de la versión A0 Dual de ρTTCN-3.

La presentación fue acompañada del paper [31] correspondiente, para el cual solicitamos y obtuvimos el apoyo de un Student Grant, esto facilitó nuestra presencia en la conferencia para la presentación en Madrid, España [17].

En el ámbito práctico el tiempo transcurrido, desde que presentamos la solución Dual hasta la escritura y presentación de esta tesis, nos permitió verficar que los resultados de la misma han sido aplicados en la industria para solucionar el problema planteado en nuestro trabajo. Al año siguiente algunos fabricantes implementaron soluciones similares en sus productos. En este sentido podemos destacar que en el año 2010 la empresa Testing Technologies, con la cual habíamos colaborado en este proceso en diferentes momentos, implementó parcialmente el acceso Dual a la plataforma de base C/C++ en un esquema similar al propuesto en la Solución Dual [39].

14.3.2. Pros

Como principal aspecto positivo del trabajo realizado, podemos concluir que la interoperabilidad automática entre herramientas y APIs de TTCN-3 en diferentes lenguajes de plataforma se ha logrado en esta implementación del prototipo para ρTTCN-3, mejorando la portabilidad de la misma según se había planteado originalmente.

El camino utilizado para realizar esta implementación es valido para extender la solución a todo el lenguaje y puede ser aplicado en el resto de la industria, de hecho ya se han dados pasos concretos en este sentido por algunas empresas.

Hemos validado además la metodología propuesta, es decir la utilización de herramientas que estén abiertas a trabajar con implementaciones y bibliotecas en ambas plataformas. Esto genera la posibilidad real de seleccionar la mejor plataforma de base para desarrollar los adaptadores necesarios en cada implementación, independientemente del lenguaje de base de la herramienta.

Con esta solución potenciamos la capacidad de reusar las bibliotecas y heramientas ya existentes para diferentes compiladores o suites de herramientas TTCN-3. También se revaloriza el trabajo realizado en diferentes campos donde TTCN-3 ha sido utilizado para desarrollar sistemas de prueba, ya que sus específicaciones podrán ser reutilizadas desde otras plataformas. Finalmente estos factores incrementan la usabilidad de TTCN-3 y promueven el desarrollo de herramientas y bibliotecas en diferentes áreas. La nueva situación generada promueve la realización de esfuerzos conjuntos entre diferentes proveedores de herramientas, en especial de aquellos que encuentren interesante el desarrollo de una plataforma abierta para TTCN-3.

14.3.3. Contras

Como un elemento a considerar desde el punto de vista de la usabilidad podemos considerar que una implementación dual es más compleja, ya que requiere tener en cuenta más elementos y diferentes herramientas. Los desarrolladores deberán tener en cuenta componentes de diferentes plataformas y deben entender la relación entre cada parte de la solución y las plataformas correspondientes. El arquitecto de las pruebas debe dominar TTCN-3, C/C++ y Java y debe decidir en cada caso cuál es la mejor opción de plataforma a usar. Basados en la naturaleza del sistema bajo prueba, el arquitecto, deberá evaluar los pros y contras de usar una funcionalidad en otra plataforma o realizar la implementación usando tecnologías nativas de la plataforma de la herramienta.

Parte III

Conclusiones Generales

Capítulo 15

Conclusiones

Los objetivos planteados al inicio de este trabajo fueron identificar y proponer soluciones a algunos aspectos de TTCN-3 que afectaban su uso, aportando de esta forma a la promoción del lenguaje, sus bibliotecas e implementaciones en las diferentes plataformas de base. Dado que nuestro trabajo se desarrolló en relación con IRISA y el proyecto Go4IT, nos interesamos especialmente en aquellos aspectos que eran relevantes en el área de testing de protocolos de red.

En ese contexto optamos por abordar las dificultades encontradas al tratar de portar definiciones de pruebas entre las distintas plataformas de base definidas en el estándar, para poder re-usar bibliotecas e implementaciones ya existentes desde diferentes herramientas en cada una de las plataformas de base.

A partir de nuestro incipiente conocimiento teórico y práctico de TTCN-3, adquirido principalmente en la construcción del compilador ρTTCN-3, incorporando el *know-how* de la importante experiencia del equipo de Go4IT en el testing de conformidad para **IPv6** y complementando estos con el estudio del estándar que define el lenguaje, pudimos verificar que el mismo es un lenguaje de testing con buena capacidad para definir casos de prueba en forma abstracta, donde logran segregarse adecuadamente los aspectos que deben implementarse en las plataformas de base. También verificamos que el lenguaje presenta ventajas específicas a la hora de realizar pruebas de protocolos de red, como su orientación a mensajes mediante el concepto de puertos, la capacidad de realizar mapeos de sus tipos de datos a los tipos de datos del sistema bajo prueba y la flexibilidad que ofrecen las interfaces definidas por el estándar para extender estas capacidades, como ejemplo de ésta característica podemos mencionar la implementación de los CoDecs del **T3DevKit** [3]. Otros aspectos que validamos en TTCN-3 fueron los de estandarización e interoperabilidad, el primero sin lugar a dudas uno de los pilares del desarrollo del lenguaje estandarizado por **ETSI** [9], mientras el segundo forma parte de los estándares definidos incorporando a estos las interfaces con la plataforma de base y los mapeos de estas interfaces a las dos plataformas que el mismo define, brindando de esta forma un mecanismo simple y consistente para implementar las adaptaciones y extensiones necesarias en las plataformas de base. También en este caso podemos citar como referencia la implementación genérica, en la plataforma C/C++, de las interfaces para la plataforma y los adaptadores del **T3DevKit**.

Es en esta área donde nuestro trabajo probó ser de utilidad para mejorar la portabilidad de las soluciones y bibliotecas existentes en las plataformas de base, como explicamos en 7.1 al momento de desarrollar este trabajo las herramientas TTCN-3 optaban por una de las plataformas sin tener en cuenta a la otra, afectando de esta forma las posibilidades de uso del lenguaje, sus bibliotecas e implementaciones.

Basados en la nueva interpretación del estándar, de que una misma herramienta TTCN-3 soportara los dos mapeos de las interfaces a las plataformas de base, diseñamos y desarrollamos un

mecanismo genérico para utilizar las implementaciones y bibliotecas de una plataforma de base desde herramientas TTCN-3 que estuvieran desarrolladas en la otra plataforma, facilitando de esta manera el re-uso de soluciones ya existentes entre herramientas nativas de diferente plataforma. Recurriendo nuevamente a los CoDecs como ejemplo, las interfaces y adaptadores del **T3DevKit**, implementados en C/C++, contarían de esta forma con un mecanismo automático para ser invocados desde herramientas TTCN-3 nativas de **Java**. Para ofrecer esta solución definimos y prototipamos la plataforma Dual para ρTTCN-3, que utilizamos luego para lograr la interoperabilidad de ρTTCN-3 con una implementación en la plataforma de base **Java** no nativa de ρTTCN-3, el cual se encuentra desarrollado en C/C++.

Esta experiencia demostró resolver el problema planteado, la nueva interpretación del estándar propuesta permitió brindar a ρTTCN-3 la capacidad de acceder de forma genérica a bibliotecas e implementaciones en la plataforma Java. Además esto se logra con costos razonables, tanto para el implementador de los casos de prueba como también para el fabricante de la herramienta TTCN-3 correspondiente, que deberá realizar las modificaciones a su producto para que este cuente con soporte Dual. La capacidad generada de usar herramientas, bibliotecas e implementaciones de TTCN-3, independientemente de la plataforma de base de cada producto TTCN-3, promueve el re-uso de estas entre diferentes productos y fabricantes, promoviendo también el desarrollo de una plataforma y soluciones abiertas para TTCN-3.

De forma complementaria, la estrategia propuesta para brindar soporte Dual a las herramientas TTCN-3 es generalizable y extensible para abordar otros problemas similares, como extender los mapeos a lenguajes no definidos en el estándar, sin necesidad de modificar este con la definición de nuevos mapeos.

En la otra parte de nuestro trabajo, el desarrollo del compilador ρTTCN-3, podemos destacar que además de servir a su fin inicial, como herramienta de experimentación para validar la Solución Dual, generó interés en otros actores de la comunidad TTCN-3 que requieren disponer de una herramienta de estas características, especialmente por su caracter libre y abierto. Queda de esta forma disponible una plataforma de desarrollo TTCN-3 para la realización de otras investigaciones y nuevos desarrollos relacionados al lenguaje.

Respecto a la experiencia de la construcción del compilador, podemos afirmar que la creación y utilización de herramientas de asistencia, desarrolladas deforma *ad hoc*, para automatizar la generación de las reglas y producciones para los reconocedores sintáctico **Bison** y léxico **Flex**, a partir del BNF provisto en el estándar, es una alternativa valida a la especificación manual de las mismas o la utilización de herramientas de terceros, especialmente para un lenguaje extenso como ρTTCN-3 y tratándose de la construcción de un prototipo como en este caso. Como mostramos en 5.1 la utilización de los asistentes significó un ahorro sustancial de tiempo para el proceso de construcción del compilador y una guía al establecimiento de una metodología de desarrollo, que posibilitó el trabajo de los equipos de forma independiente y la posterior integración de su trabajo.

El desarrollo del compilador ρTTCN-3 nos muestra también que es posible realizar un proyecto de estas características entre varios equipos distribuidos geográficamente y con importantes diferencias culturales, en este caso se logra gracias a una buena definición de objetivos, una clara distribución del trabajo y una importante tarea de coordinación y seguimiento. Esto tiene más valor si consideramos que, por su magnitud, este tipo de proyectos no puede resolverse por cada una de las partes por separado, esto es especialmente valido en el caso del **InCo**.

Por último, evaluando el trabajo realizado desde nuestro punto de vista como estudiantes, entendemos que durante el proyecto fue mayor el esfuerzo dedicado a los aspectos tecnológicos de la resolución del mismo y a la implementación de los diferentes componentes de la solución, en contraposición al esfuerzo destinado al estudio del problema planteado y las soluciones a este desde una perspectiva más conceptual que entendemos más importante. En nuestra opinión, deberíamos

haber tenido el conocimiento y práctica suficientes en la preparación del proyecto, para volcar luego la mayor parte del esfuerzo en la evalución del problema en sí, las alternativas para solucionarlo y la selección y prueba de la más adecuada de estas. Esta visión surge luego de analizar el proceso de resolución del problema para la escritura de esta tesis, en el cual surgen nuevas interpretacione de lo realizado y las opciones tomadas, no durante el proceso mismo de la elaboración de la solución donde estuvimos concentrados en los aspectos práctios, los cuales no nos permitieron adoptar esta perspectiva. Más allá de esta valoración, el haber contado con un equipo de gente con experiencia en el área temática y en las herramientas usadas, que colaboró de forma importante con nosotros en diferentes aspectos, junto a una orientación adecuada en la toma de decisiones, hizo que este factor no se convirtiese en un problema mayor para nuestro trabajo.

15.1. Resultados

Podemos valorar los resultados del presente trabajo en dos áreas diferentes, desde un punto de vista académico la presentación de los mismos en conferencias y foros dedicadas al lenguaje TTCN-3 y desde un punto de vista práctico en la utilización de los productos o los conceptos del mismo por otros actores de la comunidad TTCN-3.

15.1.1. Presentaciones realizadas

Ambos trabajos abordados en esta tesis fueron presentados en las Conferencias de Usuarios de TTCN-3 (TTCN-3 User Conferences), que constituyen el mayor foro de debate y promoción del lenguaje, en la cual participan los principales actores de la industria y la academia relacionados con este, estas conferencias de usuario se realizan de forma anual y son promovidas por ETSI como forma de impulsar el desarrollo y uso del lenguaje.

En la Conferencias de Usuarios de 2008 presentamos un trabajo sobre interoperabilidad de plataformas en TTCN-3, donde expusimos la Solución Dual: **TTCN-3 UC 2008: "TTCN-3 Tools Interoperability Between Java and C/C++ Platforms"** [31], el mismo despertó gran interés entre los participantes que se mantuvo más allá de la conferencia a través de otros foros de la comunidad TTCN-3 14.3.1.

En la conferencia de usuarios de 2010 en Beijing, China [19], presentamos un trabajo sobre el proyecto de construcción del compilador abierto y sus resultados: **"An open compiler for TTCN-3: picoTTCN-3"** [33], el mismo recibió diferentes muestras de interés de la comunidad asiática de TTCN-3 6.3.1.

15.1.2. Resultados del trabajo

Los trabajos realizados han tenido aplicaciones concretas al ser considerados por otros actores académicos o de la industria para usarlos en la solución de temas de su interés. El compilador ρTTCN-3, que en primera instancia fue usado por nosotros mismos en el prototipado de la Solución Dual, pero que desde su orígen tuvo vocación de servir a la comunidad TTCN-3 en su conjunto, despertó el interés de esta, que en diferentes oportunidades se ha comunicado para consultar por su estado y sus capacidades, con el objetivo de utilizarlo en otros proyectos. El caso más interesante es el de la empresa Huawei, que está trabajando en el desarrollo de su plataforma TTCN-3, esta empresa toma contacto con el proyecto a partir de la presentación TTCN-3: **"An open compiler for TTCN-3: picoTTCN-3."** y apartir de allí evalua la posibilidad de re-utilizar en su proyecto interno ρTTCN-3 6.3.1.

La Solución Dual fue recibida con mucho interés por la comunidad TTCN-3, al día de hoy algunos fabricantes han implementado soluciones similares en sus productos. La empresa Testing Technologies, con la cual hemos colaborado en este proceso en diferentes momentos, implementó durante el año 2010 el acceso desde su herramienta Java a la plataforma de base C/C++ en un esquema similar al propuesto en la Solución Dual 14.3.1.

15.2. Trabajos futuros

La plataforma de experimentación ρTTCN-3 que queda disponible es una herramienta que facilitará futuros trabajos de investigación y experimentación de soluciones en TTCN-3 y en sus diferentes áreas de aplicación.

Dentro de los temas de interés aún pendientes de abordar, que más nos interesan por el impacto que tiene en el área de testing de redes, podemos encontrar: la necesidad de escribir una grán cantidad de código TTCN-3 en relación con la cantidad de código fuente a testear y mejorar el sistema de matching de mensajes en general.

En el caso del soporte Dual a las herramientas TTCN-3, además de continuar aplicándose a otros productos de la industria, podría utilizarse el mecanismo propuesto para implementar mapeos a otras plataformas de base sin necesidad de modificar el estándar.

En la mejora de la portabilidad, donde nos hemos concentrado con la Solución Dual, podría avanzarse aún más en aspectos como definición de formatos binarios de distribución de bibliotecas, evitando así la obligación de distribuir los fuentes de las mismas lo cual promovería el re-uso de las mismas.

Glosario

ATS Abstract Test Suite

CBC Cipher Block Chaining

CCITT Comité Consultatif International Téléphonique et Télégraphique

CD (External) Coding/Decoding

CH Component Handling

classpath es una variable de ambiente que indica a la máquina virtual Java y a otras aplicaciones Java (por ejemplo a los instrumentos Java situados en la directory JDK1.1.x
bin) donde encontrar las bibliotecas de clase, que podrán ser entonces usadas desde algún programa.

CTMF Conformance testing methodology and framework

CORBA Common Object Request Broker Architecture

Dual en el contexto de esta tesis usamos el termino Dual o Dualización para denominar la capacidad de acceder a las plataformas de base Java y C/C++ desde una plataforma TTCN-3.

ETS Executable Test Suite

ETSI European Telecommunications Standards Institute

IEC International Electrotechnical Commission

IP Internet Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

IRISA Institute de Recherche en Informatique et SystÃ¨mes AlÃ©atoires

ISO International Organization for Standardization

IUT Implementation Under Test

jar es un archivo comprimido que incluye una estructura de directorios con Clases, lo cual permite

- Distribuir/Utilizar Clases de una manera eficiente a través de un solo archivo.
- Declarar dichas Clases de una manera más eficiente en la variable CLASSPATH.

MTC Main Test Component

PA Platform Adaptor

PCO Point of Control and Observation

PO Point of Observation

PTC Parallel Test Component

SA SUT Adaptor

SUT System Under Test

TCI TTCN-3 Control Interface

TCP Test Control Procedure

TCP Transmission Control Protocol

TE TTCN-3 Executable

TL Test Logging

TM Test Management

TMC Test Management and Control

TRI TTCN-3 Runtime Interface

TSI Test System Interface

TTCN Tree and Tabular Combined Notation

TTCN-2 Tree and Tabular Combined Notation, version 2

TTCN-3 Testing and Test Control Notation, version 3

wrapper en el contexto de la interoperabilidad de las implementaciones TTCN-3, designamos como wrapper a aquellas piezas de software que nos permiten invocar a otras piezas de software en general preexistentes, la idea es que el wrapper envuelve a la pieza original, presentando una interfaz que es requerida por otra pieza de software que precisa invocar a la origina mediante esta. En el caso que nos ocupa genralmente se tratara de implementar la interfaz del estándar para la plataforma de base del compilador.

Anexos

Anexo I: Participantes en el proyecto Go4IT

En este anexo describimos las responsabilidades de cada universidad en el proyecto de construcción del compilador y brindamos la lista de los participantes.

Universisades en el proyecto ρTTCN-3 de Go4IT

Las áreas de actividad de cada universidad fueron:

- Inco/UdelaR, Coordinación, Anlaizador Syntáctico & Lexico , Orientación Técnica
- BUPT, Traductor
- IMU, Sistema de Tiempo de Ejecucón
- INRIA, Coordinación & Revisión de Código
- ISPRAS, Testing Automático del Analizador

Equipo de colaboradores de Go4IT

La siguiente tabla enumera los colaboradores del proyecto Go4IT que participaron en la construcción del compilador ρTTCN-3.

Name	**email**	**Go4I login**
Anthony Baire	abaire@irisa.fr	abaire
Jing Chang	changj@buptnet.edu.cn	jchang
Jing Chang	changj@buptnet.edu.cn	jchang
Zhijun Ding	csdingzhj@imu.edu.cn	Dzhijun
Annie Floch	afloch@irisa.fr	afloch
Gaixia Gao	gaogaixia@sohu.com	ggao
Yan Ge	gey@buptnet.edu.cn	GYan
Xiaohong Huang	huangxh@buptnet.edu.cn	huangxh
Franck Le Gall	f.le-gall@inno-group.com	flegall
Hua Li	cslihua@rocket.imu.edu.cn	hli
Weihai Li	liweihai@gmail.com	LWeihai
Chao Liu	liuc@buptnet.edu.cn	lacking
Yan Ma	mayan@bupt.edu.cn	yma
Nikolay Pakulin	npak@ispras.ru	npakulin
Guofang Ren	gf-ren2007@yahoo.com.cn	gren
Ricardo Rezzano	rrezzano@fing.edu.uy	rrezzano
Ariel Sabiguero	asabigue@fing.edu.uy	asabigue
Cesar Viho	Cesar.Viho@irisa.fr	
Junyi Wang	wjyi@imu.edu.cn	Jwang
Wei Wang	wangw@buptnet.edu.cn	wwang
Xianrong Wang	cswxr@imu.edu.cn	xwang
Yuanyuan Wang	Betty_w00@hotmail.com	ywang
Bing Yan	ybainng@tom.com	byan
Xinming Ye	xmy@imu.edu.cn	Xye
Conghui Zhang	zhangch@buptnet.edu.cn	czhang
Yun Zhang	zhangyun@buptnet.edu.cn	ZYun
Bing Zheng	nmzhengbing210@163.com	Zbing

Anexo II: BNF de cobertura de la gramática de ρTTCN-3 vA0

El siguiente subconjunto de la gramática expresada a través del BNF del estándar define el alcance de la versión A0 de ρTTCN-3. Sin embargo los analizadores sintáctico y léxico, en algunos casos, ampliaron la cobertura por motivos de simplicidad.

```
TTCN3Module ::= TTCN3ModuleKeyword TTCN3ModuleId "{" (
ModuleDefinitionsPart) ? ( ModuleControlPart) ? "}"
( WithStatement) ? OptionalColon;
TTCN3ModuleKeyword ::= "module";
TTCN3ModuleId ::= ModuleId;
ModuleId ::= GlobalModuleId;
GlobalModuleId ::= ModuleIdentifier;
ModuleIdentifier ::= <MODULEIDENTIFIER>;
ModuleDefinitionsPart ::= ModuleDefinitionsList;
ModuleDefinitionsList ::= ( ModuleDefinition OptionalColon) +;
ModuleDefinition ::= ( TypeDef | TemplateDef | TestcaseDef ) (
WithStatement) ?;
TypeDef ::= TypeDefKeyword TypeDefBody;
TypeDefBody ::= StructuredTypeDef | SubTypeDef;
TypeDefKeyword ::= "type";
StructuredTypeDef ::= RecordDef | EnumDef | PortDef | ComponentDef;
RecordDef ::= RecordKeyword StructDefBody;
RecordKeyword ::= "record";
StructDefBody ::= ( StructTypeIdentifier) "{"( StructFieldDef( ","
StructFieldDef) *) ? "}";
StructTypeIdentifier ::= <STRUCTTYPEIDENTIFIER>;
StructFieldDef ::= ( Type( OptionalKeyword) ?;
StructFieldIdentifier ::= <STRUCTFIELDIDENTIFIER>;
OptionalKeyword ::= "optional";
EnumDef ::= EnumKeyword( EnumTypeIdentifier) "{" EnumerationList
"}";
EnumKeyword ::= "enumerated";
EnumTypeIdentifier ::= <ENUMTYPEIDENTIFIER>;
EnumerationList ::= Enumeration( "," Enumeration) *;
Enumeration ::= EnumerationIdentifier;
EnumerationIdentifier ::= <ENUMERATIONIDENTIFIER>;
SubTypeDef ::= Type( SubTypeIdentifier( SubTypeSpec) ?;
SubTypeIdentifier ::= <SUBTYPEIDENTIFIER>;
SubTypeSpec ::= AllowedValues;
AllowedValues ::= "( "( ValueOrRange ") ";
ValueOrRange ::= RangeDef;
RangeDef ::= LowerBound ".." UpperBound;
PortType ::=  PortTypeIdentifier;
PortDef ::= PortKeyword PortDefBody;
PortDefBody ::= PortTypeIdentifier PortDefAttribs;
PortKeyword ::= "port";
PortTypeIdentifier ::= <PORTTYPEIDENTIFIER>;
```

```
PortDefAttribs ::= MessageAttribs;
MessageAttribs ::= MessageKeyword "{" "}";
MessageList ::= Direction AllOrTypeList;
Direction ::=  InOutParKeyword;
MessageKeyword ::= "message";
AllOrTypeList ::=  TypeList;
TypeList ::= Type;
ComponentDef ::= ComponentKeyword ComponentTypeIdentifier
"{"(ComponentDefList) ? "}";
ComponentKeyword ::= "component";
ComponentType ::=  ComponentTypeIdentifier;
ComponentTypeIdentifier ::= <COMPONENTTYPEIDENTIFIER>;
ComponentDefList ::= ( ComponentElementDef OptionalColon) *;
ComponentElementDef ::= PortInstance;
PortInstance ::= PortKeyword PortType PortElement;
PortElement ::= PortIdentifier;
PortIdentifier ::= <PORTIDENTIFIER>;
TemplateDef ::= TemplateKeyword BaseTemplate AssignmentChar
TemplateBody;
BaseTemplate ::= ( Type) TemplateIdentifier( "( "
TemplateFormalParList ") ") ?;
TemplateKeyword ::= "template";
TemplateIdentifier ::= <TEMPLATEIDENTIFIER>;
TemplateFormalParList ::= TemplateFormalPar( "," TemplateFormalPar)
*;
TemplateFormalPar ::= FormalValuePar;
TemplateBody ::= ( SimpleSpec | FieldSpecList;
SimpleSpec ::= SingleValueOrAttrib;
FieldSpecList ::= "{"( FieldSpec( "," FieldSpec) *) ? "}";
FieldSpec ::= FieldReference AssignmentChar TemplateBody;
FieldReference ::= StructFieldRef;
StructFieldRef ::= StructFieldIdentifier;
SingleValueOrAttrib ::= MatchingSymbol | SingleExpression;
MatchingSymbol ::= AnyValue;
AnyValue ::= "?";
LowerBound ::= SingleConstExpression;
UpperBound ::= SingleConstExpression;
TemplateInstance ::= InLineTemplate;
InLineTemplate ::=TemplateBody;
RunsOnSpec ::= RunsKeyword OnKeyword ComponentType;
RunsKeyword ::= "runs";
OnKeyword ::= "on";
StatementBlock ::= "{"( FunctionStatementOrDefList) ? "}";
FunctionStatementOrDefList ::= ( FunctionStatementOrDef
OptionalColon) +;
FunctionStatementOrDef ::= FunctionLocalInst |FunctionStatement;
FunctionLocalInst ::=  TimerInstance;
FunctionStatement ::= ConfigurationStatements | TimerStatements |
```

```
CommunicationStatements | BehaviourStatements | VerdictStatements;
TestcaseDef ::= TestcaseKeyword TestcaseIdentifier "( " ") "
ConfigSpec StatementBlock;
TestcaseKeyword ::= "testcase";
TestcaseIdentifier ::= <TESTCASEIDENTIFIER>;
ConfigSpec ::= RunsOnSpec( SystemSpec) ?;
SystemSpec ::= SystemKeyword ComponentType;
SystemKeyword ::= "system";
TestcaseInstance ::= ExecuteKeyword "( " TestcaseRef "( " ") "  ")
";
ExecuteKeyword ::= "execute";
TestcaseRef ::=  TestcaseIdentifier;
ModuleControlPart ::= ControlKeyword "{" ModuleControlBody "}"
( WithStatement) ? OptionalColon;
ControlKeyword ::= "control";
ModuleControlBody ::= ( ControlStatementOrDefList) ?;
ControlStatementOrDefList ::= ( ControlStatementOrDef OptionalColon)
+;
ControlStatementOrDef ::= ControlStatement;
ControlStatement ::= BehaviourStatements;
TimerInstance ::= TimerKeyword TimerList;
TimerList ::= SingleTimerInstance;
SingleTimerInstance ::= TimerIdentifier;
TimerKeyword ::= "timer";
TimerIdentifier ::= <TIMERIDENTIFIER>;
TimerValue ::= Expression;
TimerRef ::= ( TimerIdentifier;
ConfigurationStatements ::= MapStatement;
ConfigurationOps ::= ;
SystemOp ::= SystemKeyword;
SelfOp ::= "self";
SingleConnectionSpec ::= "( " PortRef "," PortRef ") ";
PortRef ::= ComponentRef Colon Port;
ComponentRef ::= ;
MapStatement ::= MapKeyword SingleConnectionSpec;
MapKeyword ::= "map";
StartKeyword ::= "start";
Port ::= PortIdentifier;
CommunicationStatements ::= SendStatement|ReceiveStatement;
SendStatement ::= Port Dot PortSendOp;
PortSendOp ::= SendOpKeyword "( " SendParameter ") ";
SendOpKeyword ::= "send";
SendParameter ::= TemplateInstance;
ReceiveStatement ::= PortOrAny Dot PortReceiveOp;
PortOrAny ::= Port;
PortReceiveOp ::= ReceiveOpKeyword( "( " ReceiveParameter ") ") ?;
ReceiveOpKeyword ::= "receive";
ReceiveParameter ::= TemplateInstance;
```

```
StopKeyword ::= "stop";
TimerStatements ::= StartTimerStatement | StopTimerStatement |
TimeoutStatement;
StartTimerStatement ::= TimerRef Dot StartKeyword( "( " TimerValue
") ") ?;
StopTimerStatement ::= TimerRefOrAll Dot StopKeyword;
TimerRefOrAll ::= TimerRef;
TimeoutStatement ::= TimerRefOrAny Dot TimeoutKeyword;
TimerRefOrAny ::= TimerRef;
TimeoutKeyword ::= "timeout";
Type ::= PredefinedType | ReferencedType;
PredefinedType ::= CharStringKeyword |IntegerKeyword;
IntegerKeyword ::= "integer";
CharStringKeyword ::= "charstring";
ReferencedType ::= ;
TypeReference ::= StructTypeIdentifier |EnumTypeIdentifier |
SubTypeIdentifier;
Value ::= PredefinedValue | ReferencedValue;
PredefinedValue ::= VerdictTypeValue | FloatValue | OmitValue;
VerdictTypeValue ::= "pass" | "fail" | "inconc" | "none" | "error";
FloatValue ::= FloatDotNotation;
FloatDotNotation ::= <NUMBER> Dot<DECIMAL>;
ReferencedValue ::= ValueReference;
ValueReference ::= ValueParIdentifier;
OmitValue ::= OmitKeyword;
OmitKeyword ::= "omit";
InOutParKeyword ::= "inout";
FormalValuePar ::= () ? Type ValueParIdentifier;
ValueParIdentifier ::= <VALUEPARIDENTIFIER>;
WithStatement ::= WithKeyword WithAttribList;
WithKeyword ::= "with";
WithAttribList ::= "{" MultiWithAttrib "}";
MultiWithAttrib ::= ( SingleWithAttrib OptionalColon) *;
SingleWithAttrib ::= AttribKeyword AttribSpec;
AttribKeyword ::= EncodeKeyword;
EncodeKeyword ::= "encode";
AttribSpec ::= <TEXT>;
BehaviourStatements ::= TestcaseInstance |
AltConstruct;
VerdictStatements ::= SetLocalVerdict;
SetLocalVerdict ::= SetVerdictKeyword "( " SingleExpression ") ";
SetVerdictKeyword ::= "setverdict";
AltConstruct ::= AltKeyword "{" AltGuardList "}";
AltKeyword ::= "alt";
AltGuardList ::= ( GuardStatement OptionalColon) *;
GuardStatement ::= AltGuardChar( GuardOp StatementBlock);
AltGuardChar ::= "( "( BooleanExpression) ? ") ?";
GuardOp ::= TimeoutStatement |
```

```
ReceiveStatement;
Expression ::= SingleExpression;
SingleConstExpression ::= SingleExpression;
BooleanExpression ::= SingleExpression;
SingleExpression ::= XorExpression( "or" XorExpression) *;
XorExpression ::= AndExpression( "xor" AndExpression) *;
AndExpression ::= NotExpression( "and" NotExpression) *;
NotExpression ::= ( "not") ? EqualExpression;
EqualExpression ::= RelExpression( EqualOp RelExpression) *;
RelExpression ::= ShiftExpression( RelOp ShiftExpression) ?;
ShiftExpression ::= BitOrExpression( ShiftOp BitOrExpression) *;
BitOrExpression ::= BitXorExpression( "or4b" BitXorExpression) *;
BitXorExpression ::= BitAndExpression( "xor4b" BitAndExpression) *;
BitAndExpression ::= BitNotExpression( "and4b" BitNotExpression) *;
BitNotExpression ::= ( "not4b") ? AddExpression;
AddExpression ::= MulExpression( AddOp MulExpression) *;
MulExpression ::= UnaryExpression( MultiplyOp UnaryExpression) *;
UnaryExpression ::= ( UnaryOp) ? Primary;
Primary ::=  Value | "( " SingleExpression ") ";
AddOp ::= "+" | "-" | StringOp;
MultiplyOp ::= "*" | "/" | "mod" | "rem";
UnaryOp ::= "+" | "-";
RelOp ::= "<" | ">" | ">=" |"<=";
EqualOp ::= "==" | "!=";
StringOp ::= "&";
ShiftOp ::= "<<" | ">>" |"<@" | "@>";
Dot ::= ".";
Dash ::= "-";
Minus ::= Dash;
SemiColon ::= ";";
Colon ::= ":";
Underscore ::= "_";
AssignmentChar ::= ":=";
OptionalColon ::= ";
```

Anexo III: Detalles de la implementación ρTTCN-3

Se detallan algunos aspectos técnicos de la implementación del compilador ρTTCN-3, los archivos que forman parte de la implementación y las herramientas de asistencia para la generación de las especificaciones.

Archivos utilizados en la construcción del ejecutable del compilador

Los archivos que forman parte del la generación del compilador ρTTCN-3 son los siguientes

lexico.l: especificación léxica para Flex

sintactico.y: especificación sintáctica para Bison

compilador.c, compilador.h: incluye la rutina prncipal y la lógica para la ejecución del parser y las siguientes etapas del compilador, gestiona los archivos de entrada y salida

ntree.c, ntree.h: implementa el arbol sintáctico

symtab.c, symtab.h: implementa la tabla de símbolos

hashfunc.c, hashfunc.h: implementa el hash de la tabla de símbolos

functions.c, functions.h: módulo de funciones que son usadas desde los demás módulos

mensajes.c, mensajes.h: gestión de los mensajes

Makefile: archvo make para construir el ejecutable ρTTCN-3

readme.txt: indica los detalles de la implementación

Asistentes para la generación de especificaciones Bison/Flex

Las tareas realizadas para automatizar la generación de las especificaciones de Bison y Flex son las sguientes

Declaraciones BNF genera las declaraciones para Flex y Bison desde el BNF.

Producciones BNF genera las producciones con semántica básica para Flex y Bison desde el BNF y las declaraciones generadas anteriormente.

Definiciones de Nodos genera las definciones de los nodos en función del tipo del token.

Genera Acciones genera las acciones para la construcción del AST desde las producciones de Bison, utiliza las especificaciones anteriores y especialmente las declaraciones de los tipos de los Nodos y Tokens para determinar como se va cosntruyendo el AST.

Imprimir AST Genera la rutina de impresión del AST en función de los tipos de los Nodos.

Uso de ρTTCN-3

En esta sección se muestra el uso de ρTTCN-3 desde la línea de comandos para la compilación de los fuentes TTCN-3 a código ejecutable.

ρTTCN-3 -help, see the use instructions

ρTTCN-3 source.ttcn [parameters], compile source.ttcn with parameters.

Parameters: -erroroutput [filename], copy error logs to the file or stderr. -dump [filename], copy AST to the file or stdoutput. -rules, print to stdout sintatctical rules for derivation. -silent, no print, only return status (OK=0), to use to automate bulk tests.

Dependencias

Las siguientes son las dependencias para la construcción del compilador ρTTCN-3

C Compiler: gcc (GCC) 4.1.2 20061115 (prerelease) (SUSE Linux) Copyright (C) 2006 Free Software Foundation, Inc.

Bison: bison (GNU bison) 2.3 Escrito por Robert Corbett y Richard Stallman. Copyright (C) 2006 Free Software Foundation, Inc.

Flex: flex 2.5.33

Libraries: stdio.h stdlib.h

Anexo IV: Detalles de implementación Dual

El objetivo de este Anexo es describir técnicamente los cambios realizados en el package 2 de Go4IT, para la incorporación a ρTTCN-3 del soporte dual de las plataformas de base definidas en el estandar de TTCN-3.

Además se brinda la informacion necesaria para la instalacion y uso de la plataforma dualizada, la forma elegida para incorporar los cambios, permite agregar los mismos al package 2 sin generar dependencias innecesarias a la solución regular.. Esta instalación incluye el caso de prueba del DNSTester, el cual fue utilizado como experimento para la validación de la prueba de conceptos de la Solución Dual.

A continuación reproducimos el contenido archivo de liberación del parche para convertir ρTTCN-3 en una plataforma dual, los siguiente son los puntos tratados en el mismo:

1. Inicio rapido
2. Introduccion
3. Cambios Realizados en el rts de ρTTCN-3
4. Modificaciones al DNS Tester
5. Procedimiento de instalacion
6. Uso de la dualizacion
7. Alternar entre ρTTCN-3 regular y dual
8. Software de base utilzado

Inicio rapido

Se describen rapidamente los pasos para la instalacion del soporte dual a ρTTCN-3 para aquellos que ya esten familiarizados con la solucion, para quien no lo conozca o precise algun detalle deberia leer todo el documento.

1. Precondicion tener ρTTCN-3 instaldo y operativo.
2. Aplicar el patch de la solucion dual (patchp2 en pictottcn)
3. Compilar e instalar el rts dual de picottcn (libgo4it_p2_t3rts.a):

a) ejecutar runcfg en en picottcn

b) ejecutar make clean/make/make install en picottcn/rts

4. Precondicion: Seteos de ambiente para bibliotecas JNI

5. Compilar e instalar el JNI_Utils (libJNI_Utils.a): make, make install en JNI_Utils/makefiles/

6. Compilar e instalar el Dual_JNI_C_Java.a: make, make install en Dual_JNI_C_Java/makefiles/

7. Compilar e instalar el Dual_JNI_Java_C (libRTSNativo.so): make y make install en Dual_JNI_Java_C/makefile

   a) Copiar y configurar el jar TTCN3_TRI_Java.jar

   b) Construir el DNSTester dual (DNSTester) editar Makefile en DNSTester/ y setear de la variable USE_DUAL=1 ejecutar make clean/make en DNSTester/

Introduccion

En rasgos generales se crea una nueva versión del RTS (Run Time System) de ρTTCN-3, para dar soporte a las dos plataformas definidas en el estándar. Las alternativas que evaluamos para introducir los cambios en el contexto del package 2 de Go4IT fueron crear una branch en el SVN de Go4IT o realizar los cambios sobre el trunk del proyecto utilizando compilacion condicional. Optamos por esta última alternativa ya que facilita la incorporación de la platadorma dual al proyecto Go4it, minimiza el trabajo tanto para aquellos que deseen usar la solución regular como la dualizada. Se creó un script que maneja las opciones dual o no dual, seteando la variable de compilacion para dejar operativas las soluciones alternativamente.

Los cambios realizados en el rts se describen en el capÃtulo 3 e incluyen: modificaciones al rts del package2 en la biblioteca libgo4it_p2_t3rts.a, bibliotecas de soporte dual en C/C++ que dan soporte a diferentes arquitecturas y SO mediante el uso de las Autotools (Dual_JNI_C_Java.a, libRTSNativo.so y libJNI_Utils.a), bibliotecas de soporte dual en Java que esrán compilados en el archivo TTCN3_TRI_Java.jar y se instalaran a demanda cuando se requierea soporte dual y finalmente se enumeran los cambios de binarios y entorno de ejecución. En el capÃtulo 4 se explica como compilar y configurar el DNSTester para usar la implementación provista del puerto socket en Java, que permitirá ejecutar el caso de prueba del DNSTester. En el capÃtulo 5 se enumeran y describen brevemente las etapas para instalar los cambios realizados sobre el ρTTCN-3 del package 2. Para la plataforma dual, en el capÃtulo 6, se desarrolla la realización de pruebas del DNSTester, usando alternativamente la implementacion del puerto en C o en Java. En el capÃtulo 7 se detalla el procedimiento para que el DNSTester se compile alternativamente con y sin soporte dual. Finalmente se lista el software de base utilzado y sus versiones.

Cambios Realizados en el rts de ρTTCN-3

1. Cambios al libgo4it_p2_t3rts.a

2. Bibliotecas C/C++ que se agregan

3. Bibliotecas Java que se agregan

4. Cambios en binarios y ambiente

1. **Cambios al libgo4it_p2_t3rts.a)**

a) rts/T3RTSDual.h y rts/T3RTSDual.c

b) rts/T3RTSPort.h y rts/T3RTSPort.cpp

c) rts/TciCHRequired.cpp

d) rts/TciTMRequired.cpp

e) rts/TriSUTRequired.cpp

f) interfaces/tri_sa.h

g) picottcn/runcfg.sh y picottcn/rts/Makefile.am

a) **rts/T3RTSDual.h y rts/T3RTSDual.c**

1) rts/T3RTSDual.h
   - Define algunos tipos de datos que se precisan en todos lados, por ahora un par de enumerados y los prototipos de las funciones
   - se agrego el include de: “jniFunciones.h” y “triSAJava.h”
2) rts/T3RTSDual.cpp
   - se agregaron variables para guardar las referencias a los objetos Java provistos por jniFunciones
     - JNIEnv *mJNI_Env;
     - jobject mJavaTriSA;
     - jobject mJavaTriTE;

     y las funciones:
     - JNIEnv *GetJNI_Env
     - jobject GetJavaTriSA
     - jobject GetJavaTriTE
   - Declara las constantes con los nombres de las variables de environment para los paths de java y la implementacion
   - JEnvInit(), Inicializa el environment de Java y los objetos de la implementacion en el constructor de la clase
   - getPlatform(), devuelve la plataforma de un elemento del lenguaje provisto por el SA, la idea es usarlo desde el constructor del objeto y dejar seteado en este la plataforma de base donde se implementa el mismo. PlLanguage getPlatform(ElemKind eType, const TriPortId* PortId);

b) **rts/T3RTSPort.h rts/T3RTSPort.cpp**

1) agrega el include: #include ”T3RTSDual.h”
2) declara la property mPlatform que sirve para indicar la plataforma del puerto: const PlLanguage mPlatform;
3) se modifica el constructor para setear la plataforma cuando se crea una instancia del mismo: mPlatform= getPlatform(kPort, this-¿GetId());
4) agrega el metodo que devuelve verdadero si ese puerto es de plataform Java: bool IsJava() const;
5) Se dualiza el Map
6) Se dualiza el Unmap
7) Se dualiza el Enqueue

c) **rts/TciCHRequired.cpp**

- se dualiza la funcion tciExecuteTestCase para que llame a la triExecuteTestCase en ambas plataformas, dependiendo de las plataformas de los puertos que tenga, previamente carga las listas de epuertos de la tri para cada plataforma(TriPortIdList tsiPortListC, tsiPortListJ;)
- se dualiza la funcion tciReset() para que llame a ambas plataformas a la triReset, falta control de cuando hay puertos de ese lado para que llame. Duda: No alcanzaria con ver si en el archivo de configuracion hay algo para Java para ver si se ejecutan o no estas. Respuesta: no, porque la interfaz definida precisa la lista depuertos.

d) **rts/TciTMRequired.cpp**

- se inicializa el Servicio Dual y el JNIEnv

e) **rts/TriSUTRequired.cpp**

- Se modifica la funcion triEnqueueMsg mayormente para que iprima la informacion cuando es llamada desde JNI.

f) **interfaces/tri_sa.h**

- se agrega la referencia a JNI y el modificador JNICALL para la llamada a triEnqueueMsg, esto se usa para la compilacion a la vuelta del Dual. Habria otra manera de hacer esto ? con un adeclaracion explicita de la funcion que agrega algunas tareas en extensibildiad pero ademas implica que se enlace la biblioteca como estatica.

g) **picottcn/runcfg.sh y picottcn/rts/Makefile.am**

- Algunos scripts de relacionados a la generacion de binarios modificados para poder crear la solucion dual.
- Se realizaron cambios en los archivos de automake y en su invocacion para lograr la Dualizacion y mantener la independencia del ambiente de Go4it de Java.
- En el picottcn/rts/Makefile.am se agregaron los objetos nuevos, particualrmente el T3RTSDual.
- Se creo el picottcn/runcfg.sh para ejecutar el configure de picottcn con parametros de forma de incluir las bibliotecas de soporte de Java, se evita tener que usar mas de un Makefile.am con la compilacion condicional,se le agregan entonces los objetos necesarios para la Dualizacion y se corre el configure con estos parametros.

2. **Bibliotecas C/C++ que se agregan**

a) Interfaces SA y PA

b) JNI_Utils

a) **Interfaces SA y PA** Implementacion de las Interfaces SA y PA, estan en C y usan JNI para ejcutar Java desde c o viceversa. Esa implementacion esta dividida en dos proyectos, uno para la relacion C-Java y otro para la inversa.

- El proyecto Dual_JNI_C_Java produce la biblioteca libDual_JNI_C_Java.a donde se encuentran implementados los objetos triPAJava y triSAJava, en estos se implementan los metodos provided, es decir provistos por las interfaces al RTS, que invocan la llamada a la funcion correspondiente del lado de Java. Tambien se implementan las llamadas a los constructores de los objetos correspondientes del lado Java. (Agregar la lista de metodos). Para crear la biblioteca Dual_JNI_C_Java.a se compila con make (make, make clean, make install) en trunk/Dual_JNI_C_Java/makefiles

- El proyecto Dual_JNI_Java_C produce la biblioteca libRTSNativo.so donde se encuentran implementados los objetos triPlatformTE y triCommunicationTE, en este caso se implementan los metodos required, es decir provistos por el RTS a las interfaces, son un conjunto de wrappers que mediante los modificadores JNIEXPORT y JNICALL permiten ser invocados desde Java, en su codigo util extraen la informacion de los objetos Java recibidos e invocan la llamada a la funcion correspondiente del lado del RTS en C. Tambien se implementan las llamadas a los conversores de los objetos del lado Java a estructuras en C. (Agregar la lista de metodos) Para crear la biblioteca libRTSNativo.so se compila con make (make, make clean, make install) en trunk/Dual_JNI_Java_C/makefiles

b) **JNI_Utils** Implementa utilidades relacionadas con JNI, este proyecto agrupa un conjunto de funciones que permiten la utilizacion de JNI de una forma mas sencilla desde los diferentes proyectos, ocultando en gran medida las dificultades asociadas a la utilizacion de esta tecnologia.(Agregar la lista de metodos). Para crear la biblioteca libJNI_Utils.a se compila con make (make, make clean, make install) en trunk/JNI_Utils/makefiles

3. **Bibliotecas Java que se agregan**

a) TTCN3_TRI_Java Implementacion de las interfaces y el puerto socket en Java, por sencillez del despliegue se distribuye todo en el mismo jar. Este proyecto contiene la implemetacion en Java de las interfaces de TTCN-3 asi como del adaptador para el puerto del DNSTester, correspondiente a la implementacion elegida para probar. Los mismos se dividen en tri.data donde se implementan los tipos de datos necesarios para las interfaces, tri.sa donde estan las intefaces correspondientes al system adapter, tri.pa donde se encuentran implementadas las interfaces del platform adapter, por ultimo en impl.dnstester se encuentra la implementacion del adaptador del puerto socket necesario para el DNSTester. Para crear el TTCN3_TRI_Java.jar se compila con ant por fuera en: trunk/TTCN3_TRI_Java

4. **Cambios en binarios y ambiente**

a) Binarios sustituidos

b) Seteos de ambiente para bibliotecas JNI

a) **Binarios sustituidos**

1) Como resultado de los cambios descritos en los puntos anteriores se modificarion los siguientes binarios que por defecto deben recidir en usr/local/lib:
   - `file:///usr/local/lib/libDual_JNI_C_Java.a`
   - `file:///usr/local/lib/libRTSNativo.so`
   - `file:///usr/local/lib/libJNI_Utils.a`
   - `file:///usr/local/lib/libgo4it_p2_t3rts.a`
2) La biblioteca libgo4it_p2_t3rts.a sustituye la del rts e introduce relaciones con las otras bibliotecas de la implementacion dual, especificamente aquellos elementos que admiten soporte dual fueron extendidos a tales efectos. Si se extiende el alcance de la dualizacion, es decir se da soporte dual en otros tipos de elementos del lenguaje, habra que realizar extensiones equivalentes en estas clases.

3) Si bien esta biblioteca debe cambiarse cada vez que se quiera compilar el rts con o sin soporte dual, sino algunas referencias no se cumplirian, no es necesario una vez que se instalo el patch para el soporte dual deshacer este, simplemente por defecto el ρTTCN-3 seguira comportandose como antes y solo si se configura, en el archivo correspondiente, un puerto para que use plataforma Java se invocara la funcionalidad correspondiente.

b) **Seteos de ambiente para bibliotecas JNI** Para la utilizacion de JNI debe asegurarse que este instalada la API y que la configuracion del ambiente para el uso de la misma este correcta. A modo de ejemplo se podria setear el LD_LIBRARY_PATH con el camino para las biblioteca de JNI de Java, asi como todas las referencias relacionadas de Java. Estos seteos pueden realizarse desde el .bashrc.

- `export JAVA_HOME=/opt/jdk1.6.0_05`
- `export LD_LIBRARY_PATH=/opt/jdk1.6.0_05/jre/lib/i386/server`
- `export JAVA_BINDIR=$JAVA_HOME/bin`
- `export JAVA_ROOT=$JAVA_HOME/jre`
- `export PATH=$JAVA_HOME/bin:$PATH`
- `export CLASSPATH=$CLASSPATH:/usr/share/java/ant-1.6.5. jar:/usr/share/java/ant-lau jar:/usr/share/java/ant-launcher jar:/usr/share/java/ ant.jar`

Modificaciones al DNS Tester

1. DNSTester/Makefile

1. **DNSTester/Makefile** Se modifica el Makefile, se le agregan las bibliotecas necesarias para poder generar el DNSTester dual, mediante la compilación condicional con la variable USE_DUAL. De esta forma permite compilar alternativamente el DNSTester regular Go4it o el mismo con soporte dual, seteanado o no la variable USE_DUAL. Una vez decidido introducir de forma permanente el cambio para utilizar el ρTTCN-3 con soporte dual, se deberá agregar por defecto las nuevas bibliotecas a los makefile de cualquier Testcase y podra eliminarse el uso de la variable de compilación condicional.

Procedimiento de instalación del parche dual

1. Armado del patch
2. Ejecución del patch
3. Configuraciones

1. **Armado del patch** Para el armado del patch, que agregara soporte dual a la última release de ρTTCN-3 disponible en el package2 de Go4IT, se generaron con un `diff` los cambios y agregados a los fuentes y archivos de compilación para el RTS y el DNSTester, este es el archivo `patchp2`. Por otro lado se propone distribuir manualmente el archivo `jar` que contiene las implementaciones de las interfaces en Java y la implementación del adaptador para el puerto socket, contenidos en el archivo TTCN-3_TRI.jar. El comando para crear el patch es `diff -Naur package2 package2ORI patchp2` Ambos archivos quedaron disponibles para ser bajados en `www.fing.edu.uy/asabigue~/TTCN3/dual/patch`

2. **Instalación del patch dual y creación del DNSTester dual** Para la ejecución del patch deberan seguirse los siguientes pasos

   - Bajar y descomprimir el último releasae del package2 de Go4IT `svn checkout --username anonymous http://www.go4-it.eu/svnrepos/Go4IT`
   - Construir y verificar que este queda funcionando de acuerdo a su instructivo, incluyendo el DNSTester. `http://www.go4-it.org/modules/mediawiki/index.php/Technical_Documentation_Building_and_Installation`
   - Bajar el dualPatch generado y disponible en `www.fing.edu.uy/asabigue~/TTCN2/dual/patch`
   - Ejecutar en el directorio del package2 el comando patch con los siguietnes parámetros `-Np1 <file_patch>`. Ejemplo: `patch -Np1 <../../trash/patchp2 >result_aplicado.dat` verificar el resultado en result_aplicado.dat
   - Compilar e instalar el RTS dual de picottcn (libgo4it_p2_t3rts.a)
     - ejecutar `runcfg` en picottcn (se le debe dar permiso de ejecución: `chmod +rx`)
     - ejecutar `make clean` en picottcn/rts
     - ejecutar `make` en picottcn/rts
     - ejecutar `make install` en picottcn/rts
   - Compilar e instalar el JNI_Utils (libJNI_Utils.a): `make, make install` en JNI_Utils/makefiles
   - Compilar e instalar el Dual_JNI_C_Java.a: `make, make install` en Dual_JNI_C_Java/makefiles
   - Compilar e instalar el Dual_JNI_Java_C (libRTSNativo.so): `make, make install` en Dual_JNI_Java_C/makefiles. Precondición: Las variables de entorno de Java y JNI deben estar seteadas dea cuerdo al punto: 2.4.2. Seteos de ambiente para bibliotecas JNI.
   - Si algún componente da errores al compilar/enlazar puede requerirse ajustar los caminos de los include para los headers jni.h y jni_md.h, en los makefiles de los proyectos que den error.
   - Copiar el jar TTCN3_TRI_Java.jar, el cual puede bajarse de `www.fing.edu.uy/asabigue~/TTCN2/dual/patch`. El mismo debe copiarse y configurarse consistentemente en la carpeta que este especificada en el archivo de configuracion configRTS.properties (ver punto 4.4), los valores por defecto son:

     ```
     # Definiciones para la biblioteca del RTS dinamica a cargar
     lib.path=/usr/local/lib
     lib.name=libRTSNativo.so
     ```

   - Construir el DNSTester dual (DNSTester)
     - editar Makefile en DNSTester/ y setear de la variable USE_DUAL=1
     - ejecutar `make clean` en DNSTester/
     - ejecutar `make` en DNSTester/

3. **Configuraciones** Hay dos archivos de configuracion relacionados con la solución dual, uno para la infraestructura del RTS (./configRTS.properties) y el otro para la configuracion de la plataforma de cada elemento de una implementacion (./configDUAL.properties). Ambos archivos deben recidir en la carpeta del ETS.

```
./configRTS.properties
# Definiciones para el jar a cargar para el soporte java dual.
dual.path=/usr/lib/classpath/TTCN-3_TRI.jar/TTCN-3_TRI.jar
dual.sa.impl=uy/edu/fing/tri/impl/dnstester/DNSPort

# Definiciones para la biblioteca del RTS dinamica a cargar
lib.path=/usr/local/lib
lib.name=libRTSNativo.so


./configDUAL.properties
# Definiciones para la configuracion de la plataforma de los
elementos de la Test Suite

# puertos en java
port,serverPort,java
port,otroPort,c
```

Uso de la plataforma Dual

1. Configuración de la plataforma de un elemento

1. **Configuración de la plataforma de un elemento** Una vez instalado el soporte dual para ρTTCN-3 la utilización de la misma es relativamente sencilla, se trata de configurar en el archivo ./configDUAL.properties cada elemento del TE que quiera ejecutarse en la otra plataforma, en nuestro caso sera necesario configurar los elementos que se quieran ejecutar en Java, para esto se agregara al archivo una nueva línea especificando: Tipo de elemento, Identificador, Plataforma. Por ejemplo para que el puerto DNSTester utilice la implementación java provista en el jar TTCN3_TRI_Java.jar ya instalado y configurado en el putno 4.3, el archivo se vera con el siguiente contenido:

   ```
   ./configDUAL.properties
   # Definiciones para la configuracion de la plataforma de los
   elementos de la Test Suite

   # puertos en java
   port,serverPort,java
   ```

Alternar entre ρTTCN-3 regular y dual

1. Creación del RTS y DNSTester regular

2. Creación del RTS y DNSTester dual

Una vez realizada la instalación para utilizar el soporte dual en el compilador de Go4IT, se puede igual alternar entre la versión regular y dual del ρTTCN-3. Esto podría hacerse, por ejemplo, para verificar que ambos se comportan igual cuando no se dualiza ningún elemento de la implementación.

NOTA: Tengase en cuenta que si bien esta biblioteca debe cambiarse cada vez que se quiera compilar el RTS con o sin soporte dual, esto no es realmente necesario, pues una vez que se instaló el

patch para el soporte dual simplemente por defecto el ρTTCN-3 seguira comportandose como antes y solo si se configura, en el archivo correspondiente, un puerto para que use plataforma Java se invocará la funcionalidad correspondiente en la plataforma Java.

Para alternar entre estas dos implementaciones del ρTTCN-3 se pueden seguir los siguientes pasos.

1. **Creación del DNSTester comun**
   - ejecutar `configure` en picottcn/
   - ejecutar `make clean` en picottcn/rts
   - ejecutar `make` en picottcn/rts
   - ejecutar `make install` en picottcn/rts (ojo usar sudo)
   - editar Makefile en DNSTester y comentar el seteo de la variable USE_DUAL
   - ejecutar `make clean` en DNSTester
   - ejecutar `make` en DNSTester
2. **Creación del DNSTester dual**
   - ejecutar `runcfg` en en picottcn/
   - ejecutar `make clean` en picottcn/rts
   - ejecutar `make` en picottcn/rts
   - ejecutar `make install` en picottcn/rts (ojo usar sudo)
   - editar Makefile en DNSTester y setear de la variable USE_DUAL=1
   - ejecutar `make clean` en DNSTester
   - ejecutar `make` en DNSTester

Software de base utilzado

1. Compilador c: gcc v4.1.2
2. Versión de Java:1.6.0_05

Bibliografía

[1] CACTUS. http://ogi.altocumulus.org/~hallgren/Talks/Cactus/slides.html, 2001.

[2] BNF Converter. http://www.cse.chalmers.se/research/group/Language-technology/BNFC/, 2007.

[3] T3DevKit. http://t3devkit.gforge.inria.fr/, 2007. [Online; accesed 22-April-2007].

[4] ARMOR Team. Proyecto para la identificación, diseño y selección de la arquitectura más adecuada para el establecimiento de un servicio de comunicación, y el desarrollo de herramientas para lograr estas tareas. http://www.irisa.fr/armor/, 1999.

[5] Aho A.V., Lam M.S., Sethi R., and Ullman J. D. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[6] Aho A.V., Sethi R., and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 2006.

[7] MSEE MBA CSQE Douglas Hoffman, BACS. Cost benefits analysis of test automation. http://www.softwarequalitymethods.com/html/papers.html#CostBenefit, 1999.

[8] Ralf S. Engelschall. Gnu pth, posix/ansi-c based library for unix , release 2.0.7. http://www.gnu.org/s/pth/, 2006.

[9] ETSI. European Telecommunications Standards Institute. http://www.etsi.org/WebSite/homepage.aspx, 1988.

[10] ETSI. ES 201 873-1, Part 2: TTCN-3 Tabular presentation Format (TFT), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187302v030101p.pdf, 2005. [Online; accesed 28-April-2006].

[11] ETSI. ES 201 873-1, Part 3: TTCN-3 Graphical presentation Format (GFT), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187303v030101p.pdf, 2005. [Online; accesed 28-April-2006].

[12] ETSI. ES 201 873-1, Part 4: TTCN-3 Operational Semantics, Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187304v030101p.pdf, 2005. [Online; accesed 28-April-2006].

[13] ETSI. ES 201 873-6, Part 6: TTCN-3 Control Interface (TCI), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187306v030101p.pdf, 2005. [Online; accesed 19-April-2006].

[14] ETSI. ES 201 873-1, Part 1: TTCN-3 Core Language, Version: 3.2.1. http://www.ttcn3.org/StandardSuite.htm, 2007. [Online; accesed 9-April-2008].

[15] ETSI. ES 201 873-5, Part 5: TTCN-3 Runtime Interface (TRI), Version: 3.2.1. http://www.ttcn3.org/StandardSuite.htm, 2007. [Online; accesed 9-April-2008].

[16] ETSI. ES 201 873-5, Part 5: TTCN-3 Runtime Interface (TRI), Version: 3.2.1. http://www.ttcn3.org/StandardSuite.htm, 2007. [Online; accesed 9-April-2008].

[17] ETSI. TTCN-3 User Conference 2008. Madrid, Spain. http://www.mtp.es/TTCN3UC2008/home.html, 2008.

[18] ETSI. TTCN-3 User Conference 2009. Sophia Antipolis, France. http://www.ttcn-3.org/ttcn3uc2009/home.htm, 2008.

[19] ETSI. TTCN-3 User Conference 2010. Beijing, China. http://www.ttcn3-asiaforum.com/T3UC2010/en/, 2010.

[20] GNU. Autotools. http://autotoolset.sourceforge.net/, 2007.

[21] GNU. Gdb, the gnu project debugger, release 7.2. http://www.gnu.org/s/gdb/, 2010.

[22] Go4IT Consortium. Go4IT project - Advanced tools and services for IPv6 testing. http://www.go4-it.eu, 2010.

[23] Jens Grabowski and Dieter Hogrefe. Towards the third edition of ttcn. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *(TestCom 1999) Testing of Communicating Systems, Methods and Applications, ISBN 0-7923-8581-0*, pages 19–30. Kluwer Academic Publishers, 1999.

[24] Jens Grabowski, Anthony Wiles, Colin Willcock, and Dieter Hogrefe. On the design of the new testing language ttcn-3. In Hasan Ural, Robert L. Probert, and Gregor v. Bochmann, editors, *(TestCom 2000) Testing of Communicating Systems, Tools and Techniques, ISBN 0-7923-7921-7*, pages 161–176. Kluwer Academic Publishers, 2000.

[25] Guolong Wang, Huawei Technologies, Co. Ltd. TTCN-3 Development and Application In Huawei. http://www.ttcn-3.org/TTCN3UCAsia2007/Presentations/Session 1/TTCN-3 Development and Application in Huawei.pdf, 2007.

[26] Hernán Martínez. Generación automática de CoDecs para TTCN-3/Java - Extensiones Java para el T3DevKit sobre TTworkbench. TESIS DE GRADO: Extensiones Java para el T3DevKit sobre TTworkbench (Instituto de Computación - Facultad de Ingeniería - Universidad de la República) (http://www.fing.edu.uy/~asabigue/prgrado/GeneracionAutomaticaCoDecsTTCN3Java.pdf), 2007.

[27] IRISA. Institut de recherche en informatique et systemes aleatoires. http://www.irisa.fr/, 1975.

[28] ISO. Conformance testing methodology and framework. part 3: The tree and tabular combined notation. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=30621, 1998.

[29] Pablo Rodríguez Monetti. Automatizando el testing de software. www.fceia.unr.edu.ar/lcc/t523/tesina.php?campo1=9, 2009.

[30] R.S. Pressman. *Ingeniería del Software. Un enfoque práctico.* Mc Graw Hill - Interamericana de España S.A.U, 2006.

[31] Ricardo Rezzano, Ariel Sabiguero, and Cesar Viho. T3UC 2008 Paper: TTCN-3 Tools Interoperability between Java AND C/C++ Platforms. `http://www.fing.edu.uy/~asabigue/publi/rrezzanot3uc2008paper.pdf`, 2008.

[32] Ricardo Rezzano, Ariel Sabiguero, and Cesar Viho. T3UC 2008 Presentation: TTCN-3 Tools Interoperability between Java AND C/C++ Platforms. `http://www.mtp.es/TTCN3UC2008/images/papers/TTCN3 tools operability between Java and C++_URUGUAYUNIVERSITY_RicardoRezzano.pdf`, 2008.

[33] Ricardo Rezzano, Ariel Sabiguero, Frank Le Gall, Xiaohong Huang, Nikolay Pakulin, Xianrong Wang, Anthony Baire. T3UC 2010 Presentation: An open compiler for TTCN-3: picoTTCN-3. http://www.ttcn-3.org/TTCN3UC2010/June10/Paper13-An open compiler for TTCN-3 picoTTCN-3.pdf, 2010.

[34] A. Sabiguero, Anthony Baire, Alexandra Desmoulin, A. Floch, F. Roudaut, and C. Viho. Towards an ip-oriented testing framework - the ipv6 testing toolkit. `http://www.ttcn-3.org/TTCN3UC2006/FinalPresentations/TowardsIPv6TestingFramework.pdf`, 2006.

[35] A. Sabiguero, A. Floch, F. Roudaut, and C. Viho. Some Lessons from an experiment using TTCN-3 for the RIPng testing. In Ferhat Khendek and Rachida Dssouli, editors, *(TestCom 2005) Testing of Communicating Systems: 17th IFIP TC6/WG 6.1 International Conference, Montreal, Canada, May 31 - June, 2005, ISBN 3-540-26054-4*, pages 318–332. Springer, 2005.

[36] SUN. Java Native Interface. Standard programming interface for writing Java native methods and embedding the JavaTM virtual machine into native applications. (`http://java.sun.com/javase/6/docs/technotes/guides/jni/index.html`), 2006.

[37] SUN. Java Native Aaccess. JNA provides Java programs easy access to native shared libraries (DLLs on Windows) without writing anything but Java code-no JNI or native code is required. (http://jna.java.net/), 2007.

[38] Theofanis Vassiliou-Gioles , George Din and Ina Schieferdecker. Execution of External Applications using TTCN-3. http://www.springerlink.com/content/5cp0b7qpa5j6ut5a/, 2004.

[39] TTworkbench. Testing Technologies - Experts in Test Automation. (`http://www.testingtech.com`), 2000.

[40] U. Grude and F. Schröer and P. Enskonatus. TTCN-3 for .NET. TTCN-3 User Conference 2006 - 31 May to 2 June, Berlin, Germany (`http://www.ttcn-3.org/TTCN3UC2006/FinalPresentations/P6.1\_grude-presentation.pdf`), 2006.

[41] Colin Willcock, Thomas Deiß, Stephan Tobies, Stephan Schulz, Stefan Keil, and Federico Engler. *An Introduction to TTCN-3.* Wiley & Sons, 1 edition, 2005.

[42] Sergey Zelenov and Sophia Zelenova. Automated generation of positive and negative tests for parsers. `http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/z/Zelenov:Sergey_V=.html`, 2006.