TITAN: TTCN-3 tesztvégrehajtó környezet

SZABÓ JÁNOS ZOLTÁN, CSÖNDES TIBOR

Ericsson Magyarország, Test Competence Center
{janos.zoltan.szabo, tibor.csondes}@ericsson.com

Kulcsszavak: tesztelés, TTCN-3, tesztrendszer megvalósítás

Cikkünkben bemutatjuk az Ericsson TITAN nevű TTCN-3 tesztvégrehajtó környezetét, annak működését és belső felépítését. Megvizsgáljuk a TITAN fő jellemzőit és a lényeges különbségeket más, kereskedelmi TTCN-3 eszközökhöz képest. Fejlesztésünk eredményeként a TTCN-3 és a TITAN széles körben használt tesztmegoldás lett az Ericsson-on belül, emellett lehetőségünk volt részt venni az ETSI-ben folyó TTCN-3 szabványosítási munkában.

1. Bevezetés

A 80-as és 90-es években az OSI rétegmodellre épülő távközlési rendszerek tesztelésére a TTCN nyelv 2. változatát használták. Széleskörű elterjedésének gátat szabott a nyelv nehézkes táblázatos leíró formátuma, valamint a korlátozott alkalmazási terület. Ennek hatására az ETSI a 90-es évek végén a nyelv legújabb változatának, a TTCN-3-nak a kidolgozásához kezdett.

A nyelv tervezésénél figyelembe vették, hogy az eddigi konformanciateszt-alkalmazásokon felül új tesztelési módszerekre is megfelelő legyen, mint például:

– együttműködési teszt,
– teljesítményteszt,
– robosztussági teszt,
– rendszerteszt.

Az ETSI szakított a klasszikus ISO OSI modell alapú felfogással, ezzel lehetővé téve az IP alapú rendszerek, valamint a programinterfészek (API – Application Program Interface) tesztelését is.

1.1. A TTCN-3 nyelv

A TTCN-3 nyelv első szabványsorozatát 2000-ben adta ki az ETSI. Azóta több verziója látott napvilágot. A legutolsó, aktuális szabványgyűjteményt 2005-ben jelent meg [1]. A szabványosítás alatt megpróbálták elfeledni a nehézkes TTCN-2-es struktúrákat és ezzel elejét venni az ismételt negatív megítélésnek, ami a TTCN előző verziója kapcsán elterjedt a távközlési szakemberek körében. A törekvés nem volt hiábavaló, mivel egy könnyen átlátható és a tesztelést nagymértékben segítő nyelv lett az eredmény. A TTCN-3 szöveges formátuma nagyban hasonlít a széles körben elterjedt C/C++ programozási nyelvre, így sok felhasználó számára mélyebb TTCN-3-as ismeret nélkül is érthetőek a nyelvi struktúrák. A TTCN-3 nyelvről és elemeiről egy részletes összefoglaló cikkben olvashatunk [4].

1.2. A TITAN története

A TITAN kifejlesztése 2000. elején egy egyetemi diplomamunka keretében kezdődött. A fejlesztés elsődleges célja egy teljesítménytesztek végrehajtására is alkalmas, hatékony, de ugyanakkor protokoll- és alkalmazásfüggetlen futtatókörnyezet létrehozása volt. Ehhez jó alapot szolgált az ETSI-ben éppen kidolgozás alatt álló TTCN-3 nyelv.

Kevesebb, mint 1 év alatt készült el a rendszer első prototípusa, amely a nyelv lehetőségeinek csak egy részét támogatta, de belső felépítése és működése a mostani állapothoz nagyon hasonlított [5]. A TITAN azóta is folyamatos fejlődés alatt áll, követi a szabványok változásait, egyre több kényelmi funkcióval rendelkezik, és mára szinte az összes TTCN-3 nyelvi konstrukciót támogatja.

A fejlesztés főbb mérföldkövei az alábbiak voltak:

– 2000: prototípus készítése,
– 2001: párhuzamos és elosztott teszt végrehajtás,
– 2002-2003: ASN.1 nyelv támogatása, szemantikus ellenőrzéssel,
– 2004: grafikus felhasználói felület,
– 2005: teljes TTCN-3 szemantikus ellenőrzés.

2. A TITAN felépítése

A TTCN-3 fordító és futtató környezet blokkdiagrammja az *1. ábrán* látható.

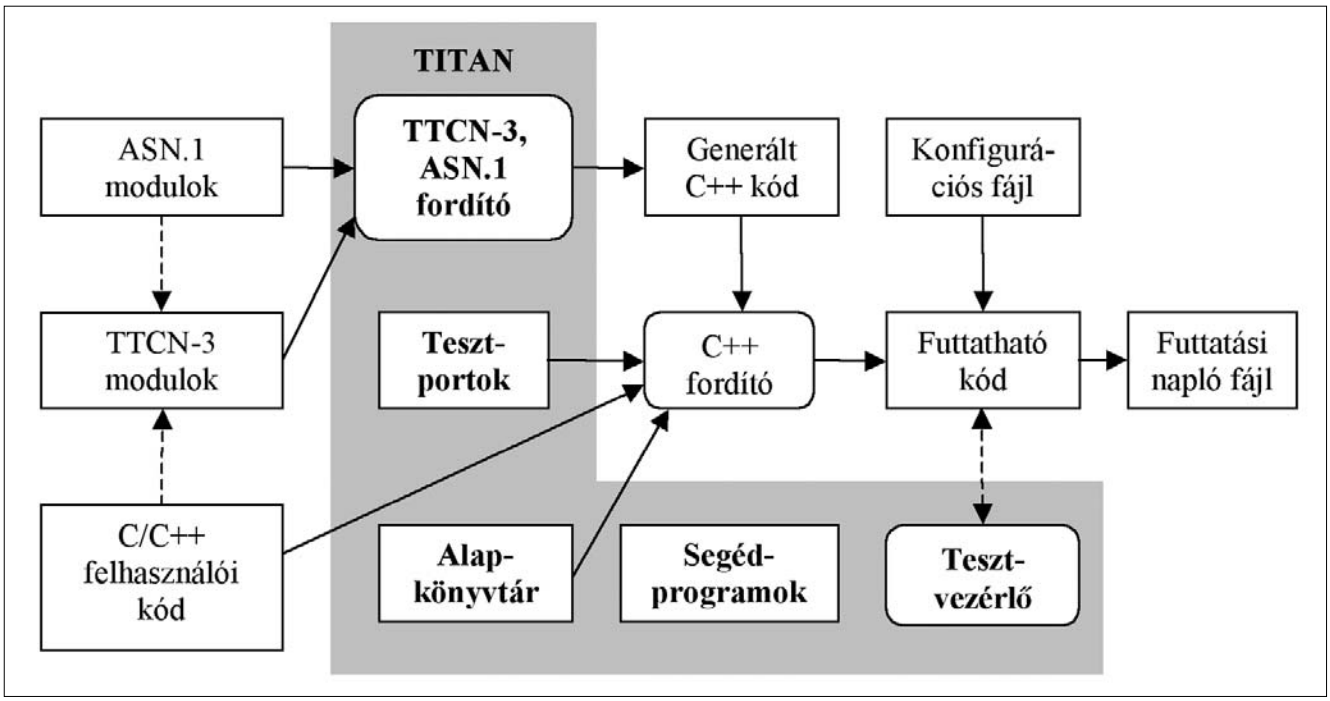
A TITAN részei a következők:

2.1. TTCN-3, ASN.1 fordító

A TTCN-3 és ASN.1 fordítóprogram a TITAN legnagyobb, legösszetettebb részegysége. Feladata a tesztkészletet alkotó modulok elemzése, ellenőrzése és a bennük található esetleges szintaktikai és szemantikai hibák jelzése. Ha a bemenet hibátlannak bizonyult, a fordító C++ nyelvű programmodulokat generál, melyek részét képezik a végrehajtható tesztsorozatnak.

2.2. Alapkönyvtár

Az alapkönyvtár tartalmazza a futtatható tesztsorozatoknak azt az állandó közös részét, amely független a tesztsorozatot alkotó modulok tartalmától. Az alap-



1. ábra A TITAN blokkdiagrammja

könyvtár kézzel megírt és előre lefordított C++ programkódból áll. Itt találhatók meg a TTCN-3 alaptípusait megvalósító osztályok, a nyelv beépített függvényeit és műveleteit (például időzítők, portok, komponensek kezelését) megvalósító funkciók. A tesztsorozat futtatásához szükséges egyéb, például a naplót készítő vagy a konfigurációs fájlt értelmező szoftvermodulok szintén az alapkönyvtár részét alkotják.

2.3. Tesztportok

A tesztportok feladata az összeköttetés és a kommunikáció biztosítása a futtatható tesztsorozat és a tesztelendő rendszer között. A TTCN-3 nyelv a külvilággal való kapcsolatot kommunikációs portokon át küldött és fogadott absztrakt üzenetekkel modellezi. A tesztport tulajdonképpen egy TTCN-3 kommunikációs port C++ nyelvű megvalósítása a végrehajtható tesztsorozatban.

A TITAN egy jól definiált programozói interfészt, úgynevezett tesztport API-t biztosít a kimenő és bejövő üzenetek kezelésére. A tesztport megvalósítások operációs rendszer hívások és általában IP alapú protokollok segítségével kommunikálnak a tesztelendő rendszerrel.

Szintén a tesztportok feladata a TTCN-3 absztrakt, struktúrált üzeneteinek átviteli csatornán használt bitfolyammá alakítása (kódolása) és visszaalakítása (dekódolása).

2.4. Segédprogramok

A TITAN-hoz tartozik még számos kis segédprogram, melyek a tesztírást, végrehajtást vagy az eredmények feldolgozását segítik. Található itt tesztfuttatást automatizáló szkript, naplófájlokat összefűző és formázó segédprogram stb.

2.5. Teszt-vezérlő

Ha egy teszteset egyszerre több párhuzamosan futó tesztkomponenst használ, akkor ezek működését össze kell hangolni. A tesztrendszer központi koordinálását a futtatható tesztsorozattól független, TITAN-hoz tartozó program végzi. A tesztvezérlő program kapcsolatban áll a tesztrendszer összes komponensével, segítségével zajlik a tesztkomponensek létrehozása és leállítása valamint a közöttük lévő port kapcsolatok felépítése és bontása.

Teljesítmény-tesztelés esetén a tesztrendszer több számítógépre is eloszthető. Ilyenkor a gépek közötti terhelésmegosztás szintén a tesztvezérlő feladata. Hogy a tesztrendszerben ne legyen szűk keresztmetszet, a tesztvezérlő kizárólag koordinátori feladatokat lát el, üzenetek továbbításában és elemi teszteseményekben egyáltalán nem vesz részt. A TITAN elosztott teszt-architektúrájának részletes leírása [6]-ban olvasható.

A tesztvezérlő biztosítja a felhasználói felületet a tesztsorozat interaktív végrehajtása közben. A felhasználói felület lehet szöveges (parancssori) vagy grafikus. Ezen keresztül a felhasználó folyamatosan képet kap a tesztrendszer pillanatnyi állapotáról és a végrehajtott műveletekről, és szükség esetén be is avatkozhat: leállíthatja az éppen futó tesztesetet vagy egy új tesztesetet indíthat el.

3. A TITAN működése

3.1. Szintaktikai és szemantikai ellenőrzés

A bemenő modulok szintaktikai elemzése a GNU flex és bison segédprogramokkal készített elemzők segítségével történik. Az elemzés során a bemenetből a fordító egy speciális memóriastruktúrát, úgynevezett ab-

sztrakt szintaxisfát épít, amely a további fordítási lépések alapjául szolgál. Az integrált ASN.1 és TTCN-3 fordítóprogram előnye, hogy a két nyelv definíciói egy közös és egységes szintaxisfába kerülhetnek. Ez megkönnyíti a TTCN-3 tesztek írását az ASN.1 leírással rendelkező protokollokhoz.

A szemantikai ellenőrzés feladata a hibás név szerinti hivatkozások, meg nem engedett műveletek, adattípus-ütközések és hasonló hibák kiszűrése. Az ellenőrző algoritmus egymenetes, tehát a szintaxisfát egyszer járja végig, de a bejárás sorrendjét a definíciók közötti hivatkozások befolyásolhatják. Szemantikai ellenőrzés során a legnagyobb kihívást a kétértelmű nyelvi elemek kezelése jelenti, ugyanis ASN.1-ben és TTCN-3-ban egyaránt előfordulnak olyan konstrukciók, melyeket a szintaktikai elemzés nem tud beazonosítani.

Ha a fordító akár szintaktikai, akár szemantikai hibát talál, az első hibaüzenet kiírása után nem áll meg, hanem folytatja tovább az ellenőrzést, hogy újabb hibákat derítsen fel. Az ellenőrző algoritmusokban hibaelfedést alkalmazunk, azaz egy létező, de hibás definícióra történő hivatkozás nem generál újabb hibaüzeneteket. Különben egyetlen egyszerű hiba is hibaüzenetek lavináját indítaná el.

3.2. Kódgenerálás

A C++ kód generálása szintén a szintaxisfa alapján történik, melyen a szemantikus ellenőrző már bizonyos egyszerűsítéseket végrehajtott, például a konstans aritmetikai kifejezéseket összevonta.

A generált kód legfőbb jellemzője, hogy statikusan tipizált, azaz minden TTCN-3 és ASN.1 adattípushoz külön C++ típus (osztály) tartozik. Ennek legfőbb előnye a futási sebességben jelentkezik, ugyanis a TTCN-3 adatértékek és mezők memóriabeli elhelyezését a C++ fordító automatikusan elvégzi.

További előny, hogy a TTCN-3 műveleteknél a típusegyezést a C++ fordító is ellenőrzi. Ez utóbbit a fordítóprogram azon régebbi változatainál használtuk ki, amelyek egyáltalán nem tartalmaztak TTCN-3 szemantikus ellenőrzést. A korai verziókban a C++ kódgenerálás a szintaktikai elemzéssel egyidejűleg történt, a szemantikai hibákra a C++ fordító hibaüzeneteiből lehetett következtetni.

A statikus típuskezelés egyetlen hátránya az összetett protokollok típusait leíró C++ osztályhierarchia nagy mérete és az emiatt megnövekedő fordítási idő. A típusok nagy kódméretét némileg ellensúlyozza, hogy a TTCN-3 értékekre, adatmintákra (templatekre) és viselkedés leírásokra (függvényekre, tesztesetekre stb.) tömör C++ kód generálható.

Dinamikus tipizálás esetén, melyet a legtöbb piacon kapható TTCN-3 eszköz alkalmaz, az adatértékeket a futtató környezet általános struktúrákból állítja össze, és minden műveletnél a típusegyezést is vizsgálni kell (például egy egész számokon végzett művelet egy általános adatstruktúrában kapja meg a paraméterét, és neki kell meggyőződnie arról, hogy tényleg egész számot kapott-e). E kiegészítő műveletek akár 10 vagy 100-szoros sebességcsökkenést is eredményezhetnek a TITAN-hoz viszonyítva.

3.3. Futtatható tesztsorozat előállítása

A teljes fordítási folyamat, így a tesztsorozat C++-ra fordítása, a generált C++ programkód és a tesztportok gépi kódra fordítása valamint a végrehajtható tesztsorozat összeszerkesztése, a make segédprogram használatával történik. A fordítási lépések közötti függőséget leíró Makefile-t a TITAN fordítóprogramja automatikusan elő tudja állítani a TTCN-3 és ASN.1 modulok, valamint a tesztportok ismeretében.

A gyakorlatban használt TTCN-3 tesztsorozatok általában sok modulból állnak. Megfigyelhető, hogy a tesztírási folyamat során a modulok nagy része (például egy protokoll üzeneteit leíró típusdefiníciókat tartalmazó modul) csak ritkán változik. Két fordítás és futtatás közötti változás általában csak néhány modult és ezen belül néhány programsort érint (leggyakrabban a teszteseteket leíró TTCN-3 utasítások változnak). Mivel egy teljes fordítás percekig, komplex tesztsorozatok esetén akár órákig is eltarthat, nem célszerű ezt a folyamatot minden apró módosítás után megismételni.

A TITAN fordítója és a make segédprogram együtt képes inkrementális fordítást végezni. Ez azt jelenti, hogy az előző fordítás részeredményeit felhasználva csak a változások által érintett modulokból generál C++ kódot, és csak a szükséges állományokat fordítja le újra. Az újrafordítandó modulok beazonosítása bizonyos esetekben nem egyszerű feladat.

Ha egy modul definícióit más modulok is használják, akkor az itt bekövetkező változások a használó modulokat is érinthetik, amelyek hatással lehetnek az őket használó modulokra, és így tovább. Ilyenkor előfordulhat, hogy egyetlen változás miatt modulok tucatját kell újrafordítani azért, hogy a végrehajtható tesztsorozat konzisztenciáját megőrizzük. Mindezek ellenére a gyakorlati példák azt igazolják, hogy a TITAN inkrementális fordítással jelentősen tudja csökkenteni a fordítási időt és ezáltal növelni a tesztsorozat-fejlesztés hatékonyságát.

3.4. Kódolás, dekódolás

A tesztfuttatás során a tesztelendő rendszernek küldött üzeneteket kódolni, míg az onnan fogadottakat értelmezni, dekódolni kell. Ennek megkönnyítésére a TITAN számos beépített kódolóval rendelkezik, melyek C++ nyelvű interfészen keresztül érhetők el. Így egy üzenet kódolása vagy dekódolása az üzenet bonyolultságától függetlenül néhány programsorban elvégezhető. Ezek a kódoló funkciók elhelyezhetők egy tesztportban vagy egy TTCN-3-ból hívható, de C++ nyelven megírt külső függvényben.

Az ASN.1-ben leírt üzeneteket a TITAN a BER szabályai szerint képes kódolni, a TTCN-3 adattípusokhoz kétféle, egy szöveges (TEXT) és egy táblázatos, bit alapú (RAW) kódolás létezik. TTCN-3 típusok esetén a kódolási szabályokat, melyek lehetnek egészen összetettek is, attribútumok segítségével lehet megadni.

3.5. Grafikus felület

A TITAN-hoz a tesztvezérlővel egybeépített grafikus felület is tartozik, mely a TTCN-3 tesztsorozatok fejlesztésénél és futtatásánál is segítséget nyújt a felhasználók számára.

A *2. ábrán* a grafikus felület fő ablakának képernyőképe látható. Bal oldalon találjuk a tesztsorozatot alkotó modulok, tesztportok és egyéb fájlok listáját. A jobb oldali ablakban pedig a fordítás menetét követhetjük nyomon, – most épp egy inkrementális fordítás kimenete látható.

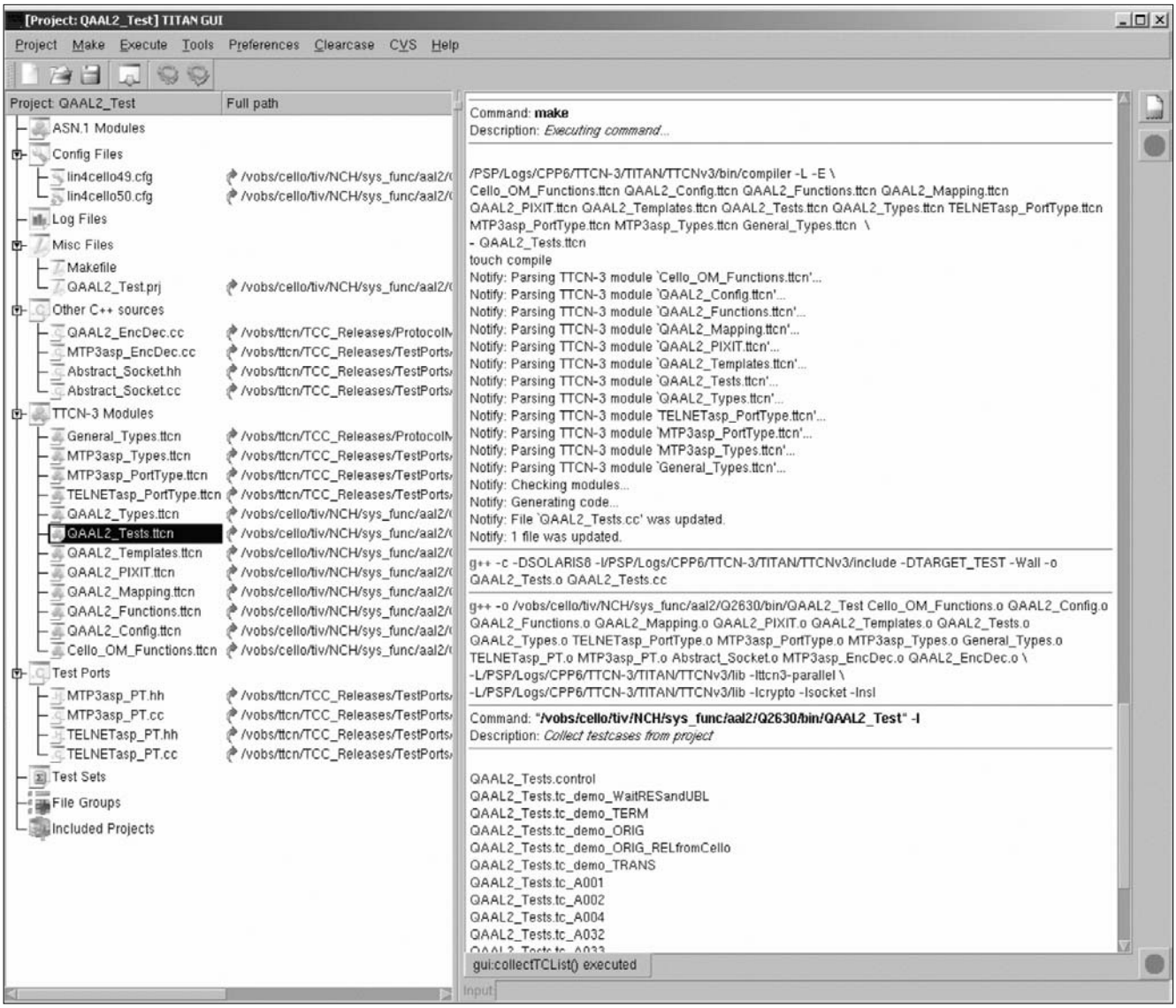
4. TTCN-3 interfészek

Egy futtatható TTCN-3 tesztsorozat a külvilágot interfészeken keresztül érheti el. Az ETSI két szabványban (TTCN-3 Runtime Interface, TRI [2] és TTCN-3 Control Interface, TCI [3]) hat ilyen interfészt definiált. A TRI két interfésze a tesztsorozatnak a tesztelendő rendszerrel illetve az operációs rendszerrel való kapcsolatát (pl. időzítés) írja le. A TCI négy interfészből áll: tesztmenedzsment (tesztesetek futtatása és a tesztsorozat paraméterezése), kódolás és dekódolás, tesztkomponensek kezelése és naplózás.

A szabványok programozási nyelvektől független adattípusok és eljárások segítségével írják le az interfészeket, és ezekhez C, JAVA és néhol XML nyelvű leképzést adnak. Az interfészek szabványosításának célja, hogy az ezeknek megfelelő TTCN-3 futtatókörnyezetek egymással csereszabatosak legyenek.

A TITAN a fenti szabványos interfészek közül jelenleg egyiket sem támogatja. Ennek több oka van: egyrészt amikor az interfész-szabványok 2002-ben és 2003-ban megjelentek, akkor a TITAN már egy kész, kiforrott rendszer volt a saját interfészeivel. Másrészt a fejlesztés során más szempontokat részesítettünk előnyben, mint a szabványosítók. Elsődleges célunk az volt, hogy egy teljes, működőképes és hatékony tesztrendszert állítsunk elő úgy, hogy a felhasználónak a lehető legkevesebb külső programmodult kelljen a rendszerhez hozzáfejlesztenie. Így a TITAN egyedül a tesztelendő

2. ábra A grafikus felület



rendszer felé biztosít programozói felületet, ez pedig a tesztport-interfész. A TRI és TCI összes többi funkcióját a TITAN beépített módon támogatja, ezekhez nem biztosít programozói hozzáférést.

A TITAN tesztport interfésze több szempontból is előnyösebb a TRI-nél. A tesztportok mindig egy TTCN-3 kommunikációs porthoz (és ezáltal egy protokollhoz) kapcsolódnak, így az egyidejűleg tesztelt többféle protokoll szétválasztásáról az interfész maga gondoskodik.

TRI használata esetén a tesztelendő rendszer felé irányuló összes üzenet egyetlen függvényhíváson és a felhasználó által írt, úgynevezett adapter modulon halad keresztül, és a szétválogatást itt kell megoldani. Ha a tesztelésbe egy újabb protokollt vagy rendszer-interfészt akarunk bevonni, akkor ez a TITAN-ban építőkocka elven egy új tesztport hozzáadásával könnyen megoldható, míg TRI esetén ugyanez az adapter újratervezését jelenti. A tesztport interfész elosztott teljesítménytesztek esetén is kedvezőbb, ugyanis a TTCN-3 párhuzamos teszt komponensekhez kapcsolódó tesztportok nem okoznak szűk keresztmetszetet.

A szabványos interfészek utólagos beépítése bizonyos esetekben nehézséget is okozhat, mert ezek a TITAN-nal ellentétben dinamikus tipizálást és többszálú működést feltételeznek. A TRI és TCI interfészei általában nem a hatékony működést preferálják, így ezek gyakorlati haszna is megkérdőjelezhető a TITAN-ban. Használatukkal többnyire nem lehetne a TITAN beépített funkcióihoz képest gyorsabb, egyszerűbb vagy kényelmesebb megoldásokat előállítani.

5. Összefoglalás

A TITAN fejlesztése és használata révén az Ericsson is bekapcsolódott az ETSI TTCN-3 szabványosító munkájába. Aktivitásunkat jól mutatja, hogy a nyelv szabványához eddig összesen beküldött 340 db módosító javaslat (Change Request) több mint fele (196 db) az Ericssontól származik. Ezek többsége a nyelv leírásában talált kétértelműségekre vagy ellentmondásokra ad feloldást, de számos olyan nyelvi kiterjesztést is javasoltunk, melyek a TTCN-3 használatát egyszerűbbé, kényelmesebbé teszik.

A TITAN 2003. óta hivatalosan elfogadott és támogatott teszt eszköz az Ericssonban. Azóta az Ericsson Magyarország Kft-ben egy közel 40 fős részleg foglalkozik a TITAN valamint a rá épülő TTCN-3 teszt megoldások (tesztportok, protokoll modulok és kész tesztsorozatok) fejlesztésével és karbantartásával. Megrendelőink az Ericsson termékfejlesztő részlegei a világ minden tájáról.

Bár a TITAN a cégen kívül piaci forgalomba nem került, így is jelentős és folyamatosan növekvő felhasználói táborra tett szert az elmúlt évek során.

Közel 50 tesztport és csaknem 100 protokoll modul készült eddig a TITAN-hoz, mely lehetőséget nyújt a távközlési rendszerek széles spektrumának teszteléséhez.

Irodalom

[1] ETSI ES 201 873-1 V3.1.1 (06/2005)
The Testing and Test Control Notation version 3.
Part1: Core Language
[2] ETSI ES 201 873-5 V3.1.1 (06/2005)
The Testing and Test Control Notation version 3.
Part5: TTCN-3 Runtime Interface (TRI)
[3] ETSI ES 201 873-6 V3.1.1 (06/2005)
The Testing and Test Control Notation version 3.
Part6: TTCN-3 Control Interface (TCI)
[4] Szabó János Zoltán:
A TTCN-3 tesztleíró nyelv,
Magyar Távközlés, XII. Évf., 2. szám, 2001. február
[5] János Zoltán Szabó:
Experiences of TTCN-3 Test Executor Development,
Testing of Communicating Systems XIV,
Application to Internet Technologies and Services,
Edited by Ina Schieferdecker, Hartmut König and
Adam Wolisz, Kluwer Academic Publishers, 2002.
[6] János Zoltán Szabó:
Performance Testing Architecture for
Communication Protocols,
Periodica Polytechnica, Electrical Engineering,
Budapest University of Technology and Economics,
2003. 47/1-2.