Testing of switch node with Call Generator software module

Boštjan Vlaovič, Zmago Brezočnik
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova ul. 17, SI-2000 Maribor, Slovenia
e-mail: {bostjan.vlaovic,brezocnik}@uni-mb.si

Abstract

This paper introduces development of software for the IskraTEL's switch node SI2000 V5 — MLB module. A Software Call Generator module (CG) has been developed for easy and thorough testing of software developers' program code. Until now, developers had to use external testing equipment. There are not enough test machines for every developer to use, due to their high cost. They are in the hands of validation and testing personnel most of the time.

One of the objectives was also geographical independence of CG users. CG can be used from any place in the world. The only requirement is Ethernet connectivity with the switch node that we want to test. This enables testing of already installed switch nodes in Russia, Turkey, etc. We are not limited to testing areas in the company. CG can be also successfully used by the service department. Any WWW browser can be used for the user interface. This way operating system independence is guaranteed. The paper presents planning and development of CG and a practical illustration of its use.

Key words: telecommunications, testing, SI2000, SDL, call generator, switch node

Introduction

Bell's invention of the telephone in 1876 had one of the greatest impacts on mankind. Since then telephone network has been growing each year. Nowadays nearly a billion telephones exist worldwide, reliably enabling communication from nearly any point on earth with any other point. In the last decade we are witnessing a huge growth in telecommunications market, mostly due to the deployment of Internet. Very demanding markets expect telecommunication companies to develop and implement all new technologies in the shortest possible time. Therefore development and implementation of new technologies were shortened in the past years considerably. A very important part of every development is validation and testing.

Validation is a systematic evaluation of product functionality through its development process [1]. Reviews and tests are performed at each phase of the development process. To ensure that the product requirements are complete and testable, testability should be reviewed at the earliest stage possible. The two major types of validation activity are:

1. Reviews: Reviews include "walkthroughs" and are actually a form of inspection, rather than testing. Reviews include reviews of documentation to make sure that the documentation will support operation, maintenance and future enhancements.

2. Testing: Unit testing, progressive levels of integration testing, validation testing and acceptance testing each serve a different purpose. Weaknesses in any testing phase will compromise the integrity of validation and acceptance tests.

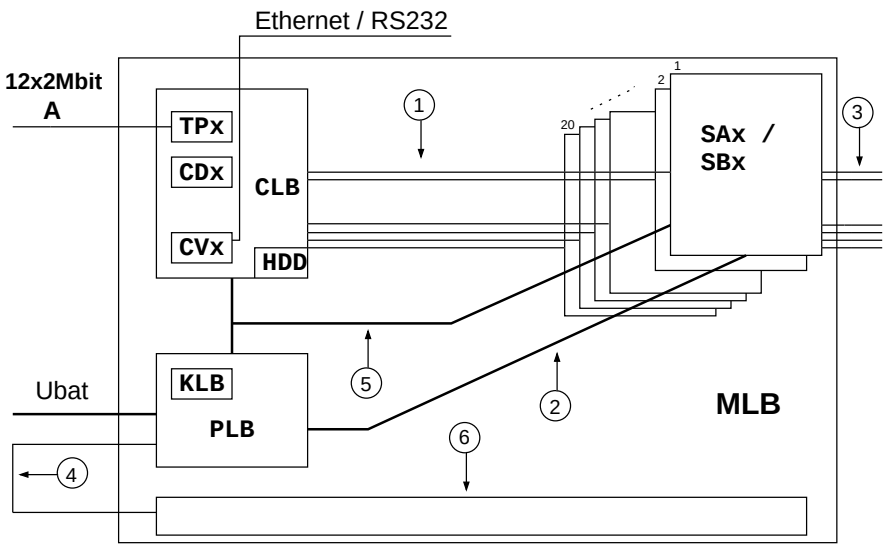
We will focus on testing of a new software functionality in the earliest stages of its development. Our goal was to enable testing to every developer. Although modern switch nodes are basically computers, most of the testing is still done with a black box approach. Testers usually use expensive external test equipment.

CG software module introduces a different approach to testing of switch nodes. It can not fully replace other test methods, but it provides a cheap, quick and reliable test environment for the developer of software code.

First we will take a quick look at digital switching system SI2000 V5. That will give us enough understanding about the basic concepts and building blocks of the hardware and software. After this introduction we will explain how and why we decided to implement CG the way we did. In the final section of the paper we will provide a practical example of its use through a real-life situation.

Digital switching system SI2000

IskraTEL d.o.o. is the company for development, marketing, planning, manufacturing, installation and servicing of telecommunication systems. It was formed in 1989 with the capital from Slovene investors and the German company Siemens AG [2] and is the largest Slovenian telecommunications company. The new generation of its digital switching system is called SI2000 V5. It is a digital switching system with a few hundred to several thousands of ports. It



CVx - VME processor unit
CDx - Communications Controller, version x
MLB - Line Module, version B
PLB - Power and RC unit, version B
KLB - Measurement unit, version B
CLB - Line Module Controller, version B
SAx - Analog Subscriber Unit, version x
SBx - Basic Rate Access Subscriber Unit, version x
TPx - Symmetrical Primary Rate Access Interface, version x
Ethernet / RS232 - Local connection for operation and maintenance
A - Interface for 2Mb/s link in compliance with ITU-T G.703 and G.704

1 - Internal communication links
2 - Ringing
3 - Subscriber / trunk lines
4 - Fan control
5 - Power supply
6 - Fan unit + filter

HDD - Hard Disk Drive
Ubat - Battery feeding (48V)

Figure 1: Line Module

is an advanced modular system that offers basic functionality as well as a rich range of services including ISDN, signalling No. 7, Centrex, ATM, IP, etc. Since CG software module is developed only for analog subscriber line[1] testing, the paper will focus on MLB (Line Module Version B) (Fig. 1) with SAx (Analog Subscriber Unit Version x).

The MLB makes the core of the access node (AN). The task of MLB is to connect analog subscribers, ISDN terminals and network transmission paths [3]. The line module performs the tasks of an access network, including the detection of calls, subscriber ringing and performance of measurements. Sub-rack configuration of MLB consists of mechanical parts, a back panel and plug-in units (PLB, CLB, SAx and SBx)(Fig. 1). The back panel has 24 slots. The first two slots are reserved for CLB, the second two for PLB and the remaining positions for 20 peripheral units. All connectors except the CLB connector are identical. Therefore the layout of the units in the system sub-rack is optional. The CLB is the basic plug-in unit of the line module. It incorporates TPx, CVx, CDx, HDD (Fig. 1). Symmetrical Primary Rate Access Interface (TPx) unit provides connection to the digital network (ISDN primary rate access (PRA), V5.2i interfaces). For better understanding of this paper we only need to understand the functions of CVx, CDx and SAx. CVx is connected to individual functional groups of CLB via an adapter containing VME (Virtual Memory Extender) and a local bus. CVx performs the following access node functions:

- controls and monitors MLB,
- provides 2 Mb/s links for the connection to the central module,
- controls the switching network and is connected to peripheral units via serial links,
- all applications run in the pSOS+ (Plug-in Silicon Operating System) providing the correct scheduling of tasks to the various calls,
- communication with local management terminal (MT) via an RS232 or Ethernet bus,
- control of hard disk via the SCSI (Small Computer Standard Interface).

CDx is used for scanning analog subscribers, sending tones to the user in the event of congestion, V5.2i or switch node (SN) failure, for super-frame synchronisation, processing of conference calls and data communication using the High-level Data Link Control (HDLC) protocol (32

[1]The same principle can be used for the development of CG for SBx (Basic Rate Access ISDN Subscriber Unit).

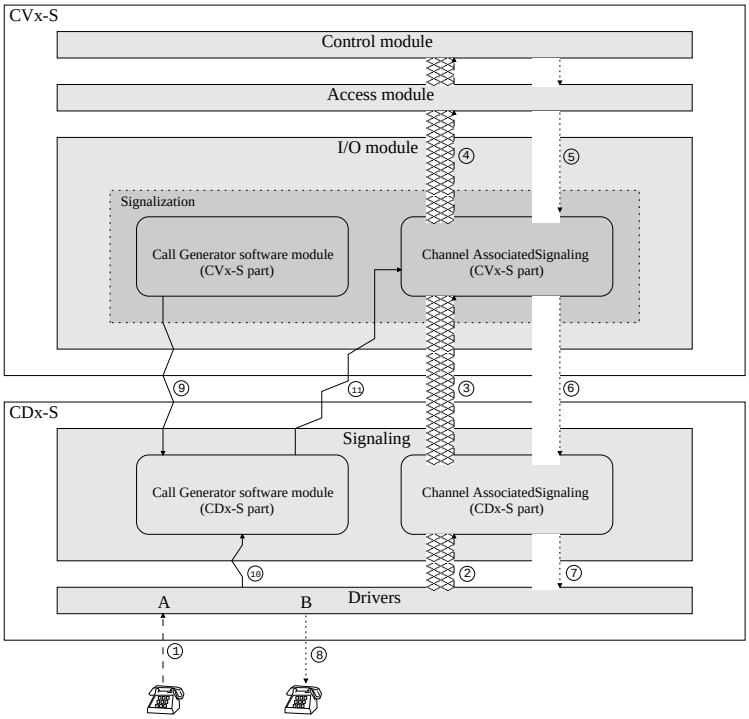


Figure 2: Software hierarchy of SI2000 and analog subscriber signal path.

Digital Subscriber Signalling System No. 1 (DSS1) controllers).

Peripheral units are inserted in the MLB sub-rack starting from the left to the right after the central units. In the module 20 slots are available for the peripheral plug-in units. They are connected in a form of a star, thus minimizing the mutual influence among the plug-in units and allowing plug-in unit replacement under voltage. The interface on each peripheral unit used for the connection with CLB has the following characteristics:

- 16 Mb/s transmission rate,
- bidirectional 4-wire transfer (clock, frame, bus up, bus down),
- 8-bit data bus,
- protocol on the bus is of master-slave type.

The peripheral unit SAx serves for two-wire (a/b) connection of analog terminals. There are 32 analog subscriber line circuits on the plug-in unit. The unit has been designed for the application in public systems.

SI2000 software distribution

The core of the modern switching system is its software code. It provides the functionality, control and management for the system. Software is divided in system and application part. The following SI2000 system software is used by CG:

- pSOS+ real-time operating system,
- TCP/IP protocol stack,
- Real-time database management system (RTDBMS) [4] that includes:
  - mechanisms providing the applications software with the possibility of accessing the database,
  - SQL (Structured Query Language) server.

The system software is executed in real time. This means it responds to the events in the environment in the predefined times. The execution of software code is divided between CVx and CDx modules. Fig. 2 describes software hierarchy of SI2000 switching system. We decided to put CG code on both parts of the system. The CVx-S[2] part will do all the management and control, have access to the database and provide access to hard disk. Because we want to cover as much application software with our test module as possible and we don't want to change any existing software, generation of calls (generation of call signalling) should be done as close to the usual origin of call signalling as possible. As you can see in Fig. 2 telephone lines are connected to CDx part of switch node. That is why part of CG naturally belongs in the CDx-S. Analog subscriber signal path for simple call from port A to port B is shown in MSC [5] diagram in Fig. 3.

[2]Software code that is in the CVx-S is actually processed with the CVx module. The CDx-S includes code that is processed with CDx module.

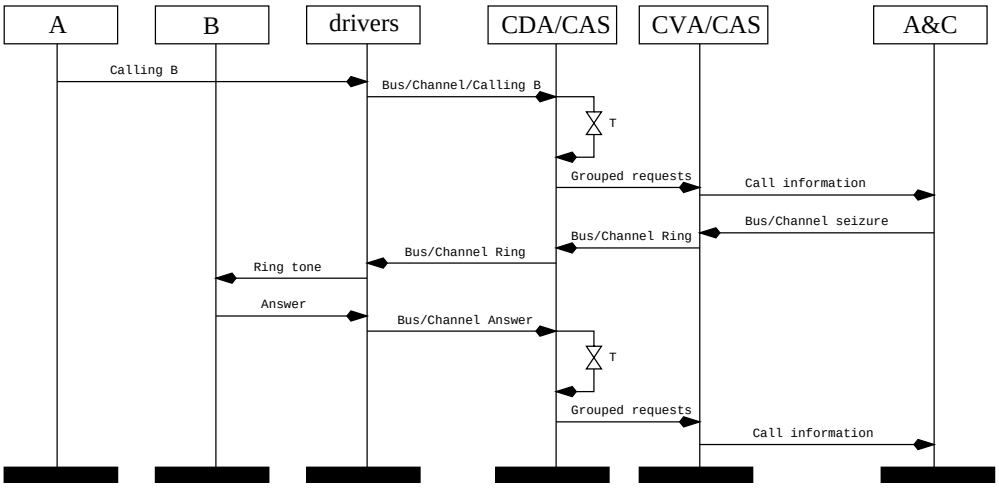


Figure 3: Simplified MSC diagram of telephone call from port A to port B.

If we wanted CG to work, we had to send all signals from low level driver to it (signal route 10 — Fig. 2). Additionally we had to copy all signals that were coming from the Control&Access modules (signal route 5 to signal route 10 — Fig. 2). This was accomplished with a minor change in the driver code. With all needed information at our disposal we were able to write CG that can simulate incoming calls, intercept all outgoing actions and connect the two or more, in the case of conference call, ports that we want to test. Code is written in Specification and Description Language (SDL) [6, 7, 8, 9].

Call Generator software module

CG consists of two main parts: the management part (CGman) and the call generation part (CGcg). Our chosen architecture had to provide:

- control of CG through TCP/IP network,
- test scenario in the database,
- SQL database control over TCP/IP network,
- minor changes in existing software code,
- a low microprocessor load.

If we wanted to use TCP/IP functionality and database connectivity, we had to put management part of CG (CGman) on CVx-S, since CDx-S does not have access to the hard disk. CGman is one SDL process. It can read test scenario from the database, package and send all the needed information to the CGcg, receive call status information from every test telephone process (TT) and manage CG. CGcg part is located on CDx-S side and includes a management process (TTman) and any number of test telephone processes (TT). TTman receives signals from CGman and switch node driver. It checks the received packet header, finds the destination TT and forwards received information. Grouping of call signalling for TTs is also its job. TTs are individual and independent entities, just like regular telephones that are connected to the switch node. TT can simulate all analog telephone signals that would normally be generated by the driver. TTman forwards all TT's signals to the CAS (CVx-S part) every 4 ms (T in Fig. 4). Grouping of signals minimizes the system load. It is usually done in CAS (CDx-S part). We wanted to keep modification of the existing software to the minimum. Hence we decided that CG will do grouping of signals for the virtual telephones. It does it in the same way as CAS (CDx-S part). CAS (CVx-S part) that receives grouped signals does not notice the difference between the two. Fig. 4 shows a signal path in the case of call generation with the CG. If you compare Fig. 3 and 4, you will notice the described differences. CG test scenario is located in the database. CG uses three database tables. The first table is used for the CG management information, the second holds data that defines TTs and the third includes actions for every TT. Most of database records will be explained through the practical example based on two virtual telephones. TT-A will be calling TT-B. Let us assume that telephone number of TT-B is 2354. CG user first has to prepare SQL[3] entries that describe the general behaviour of CG. This information is stored in the first table. The user can set start and stop date/time for the given test scenario. If the user chooses to repeat the same scenario any number of times, he/she can set the restart period. For each restart of scenario we can write traffic data to the hard disk. At this point we can also reset traffic data. That way we can repeat the same actions any number of times and compare the stored results during the analysis phase. For greater flexibility of test scenario we decided to use our own time unit (`action_pause_unit`). That way we can change timings of scenario very quickly. The remaining two records keep information about the number of starts of CG and the date/time of the last start. We need this

[3]For communication over TCP/IP we use `irtsql` that is part of the SI2000 system software.

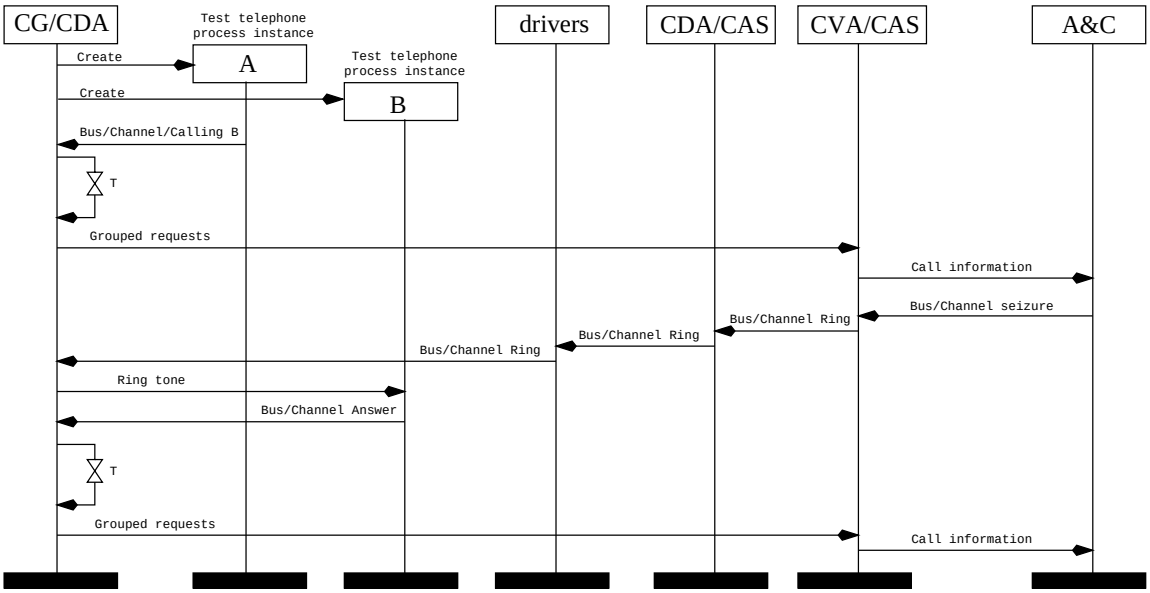


Figure 4: Simplified MSC diagram of telephone call from port A to port B when using Call Generator.

information for the easier problem location if something goes wrong and switch node has to be reset. If we want to control CG in real time, we can use TCP/IP network, too. Switch node's system software supports Common Gateway Interface (CGI). Through CGI we can control the starting/stopping of CG, reset traffic data and refresh CG data from the database.

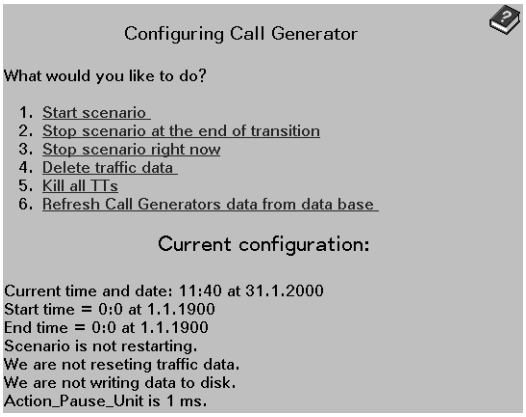


Figure 5: WWW browser interface to CG control.

The next step is setup of virtual telephones. Each TT is identified by its (bus, channel) number pair. `Call_duration` is used for arbitrary interrupts of call in progress (i.e. between the group of signals that represents dialled zero). To simulate random behaviour of TTs, we introduced additional variables. The first pair (`call_duration_h, call_duration_l`) represents the upper and lower limit for the duration of call. To be able to define calls with random behaviour in more detail we introduced another two pairs. The first represents the upper and lower limit for the random generation of pause between dialling (`short_pause_h, short_pause_l`). The second one is used for the generation of random time length for "conversation" (`long_pause_h, long_pause_l`). For every TT we keep track of all outgoing and incoming calls (`num_out_calls, num_succ_dials, num_in_calls, num_acc_calls`). Each TT can act as the source of a call — it can dial other TTs — or answer an incoming call. In the last database table a user can describe behaviour of each TT for the outgoing (`out_action, out_pause`) and incoming (`in_action, in_pause`) calls. Each set of actions is identified by `tt_id`. For the normal operation of TTs we need 21 actions (Table 1).

Table 1: Actions for TT control.

No.	Code	Action
1-10	0-9	numbers from 1 to 10
11	11	*
12	12	#
13	33	*RR*
14	20	*off-hook*
15	30	*on-hook*
16	40	*GND on*
17	50	*GND off*
18	66	*end of selection*
19	99	*end of scenario*
20	80	*no operation (NOP)*
21	100	*wait for ring*

We will focus on the table that describes TT's actions. Table 2 shows the needed entries in actions table to place a call from TT-A (`tt_id=1`) to TT-B (`tt_id=2`).

As you probably noticed TT-A won't accept incoming calls, since its first `in_action` is 99 (*end of scenario*). Its mission is to call TT-B, "talks" with it for 1500 units

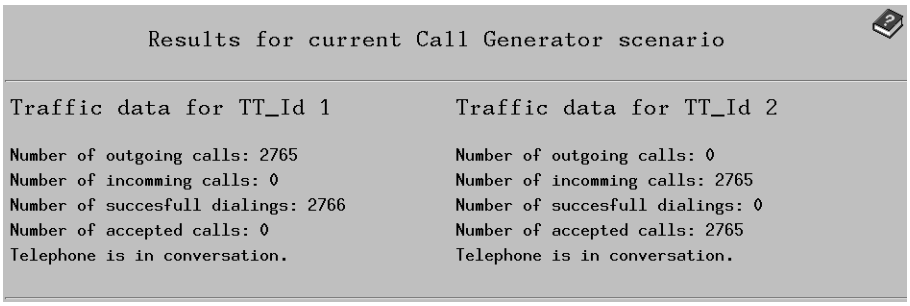


Figure 6: Traffic results of individual TTs in the WWW browser window.

(`action_pause_unit`[4]) and repeat the procedure after 200 units of pause. Action *end of selection* helps us with the call status tracing. The other test telephone (TT-B) will be only accepting incoming calls. Its first action tells it to accept every incoming call after two rings.

Table 2: Scenario for call from TT-A to TT-B.

tt_id	No.	in_action	in_pause	out_action	out_pause
1	1	99	NULL	20	300
1	2	NULL	NULL	2	100
1	3	NULL	NULL	3	100
1	4	NULL	NULL	5	100
1	5	NULL	NULL	4	100
1	6	NULL	NULL	66	1500
1	7	NULL	NULL	30	100
1	8	NULL	NULL	99	100
2	1	100	2	99	NULL
2	2	20	500	NULL	NULL
2	3	30	100	NULL	NULL
2	4	99	300	NULL	NULL

In this command we made an exception and `in_pause` does not represent time units. There is also another similar exception. If a user chooses to use randomly generated calls, he/she can use reserved numbers to tell that to CG. Number 0 will generate a pause in the limits defined by `short_pause` pair and number 1 will choose randomly generated number in the limits of `long_pause` pair. We will not be using randomly generated numbers in our example. After TT-B "picks up" the phone, it waits or "talk" to the TT-A for 500 time units. It is able to receive new calls 400 units after call termination. This is the simplest example available. With this design the user is not limited by CG in any way. He/she can describe any real situation. With the mid-call termination (`call_duration`) he/she can simulate other, unexpected problems and test the switch node's response.

When the switch node is reset or database tables are refreshed, CGman reads the information from the database, packages it in small packets and transmits it to the TTman with a simple handshake protocol. TTman evaluates received data and creates described TTs. In our example it would create two TTs (TT-A and TT-B). Each TT stores its own actions and becomes an independent functional unit. Every TT status change is reported to CGman. It stores all statistical information about every TT's action. A user can view the results on-line with his/her favourite WWW browser (Fig. 6) or check records in the database. On-line user interface also keeps track of cumulative statistics (Fig. 7), so user can quickly notice unexpected behaviour.

Every TT's action is forwarded to TTman. It stores all actions for the period of 4 ms. This is also the smallest time unit for any action to be performed in the real time environment. TTman packs all received actions in a predefined packet which is in 4 ms time intervals sent to CAS (CVA-S part). At this point we would like to emphasise that none of the ports that are not part of the test scenario are effected. Driver and TTman perform their packaging independently and in parallel. All other ports can be used while CG is working. Even the tested ports can be used, since switch node is not aware of CGs existence. If you connect telephones to the same ports as in the test scenario (TT-A, TT-B), the telephone connected to the same port as TT-B will be actually ringing. When the scenario defines that TT-A and TT-B are in "conversation", users can pick up the real telephones and talk to each other — therefore the connection is not virtual, it is true connection and all switch node's software and hardware is involved in the same way as with external testing devices. The only hardware part of SN that can not be tested is the telephone-SN connection since the calls are generated inside the SN. Another pitfall compared to external testing devices is line status checking. External testing devices can check if the correct ports have been connected. Usually they use a set of fixed frequencies to do that. The modular organisation of SI2000 V5 provides us with means to do line checking too, but since this would require additional changes of the existing software, we decided to implement this functionality at a later time if it proofs to be necessary.

When Control&Access modules seize the required channels and send ring signal to TT-B through CAS processes and driver, all signals are copied to TTman. It filters the received signals and checks if they are destined for any of the TTs. If the signal is found to belong to one of the TTs, TTman forwards it to the TT in question. All other signals are discharged. This is how CG works. Most of the code is written in SDL. When features (CGI, database access, etc.) were not possible to realize in SDL, we wrote C

[4] User can define `action_pause_unit` in 1 ms increments.

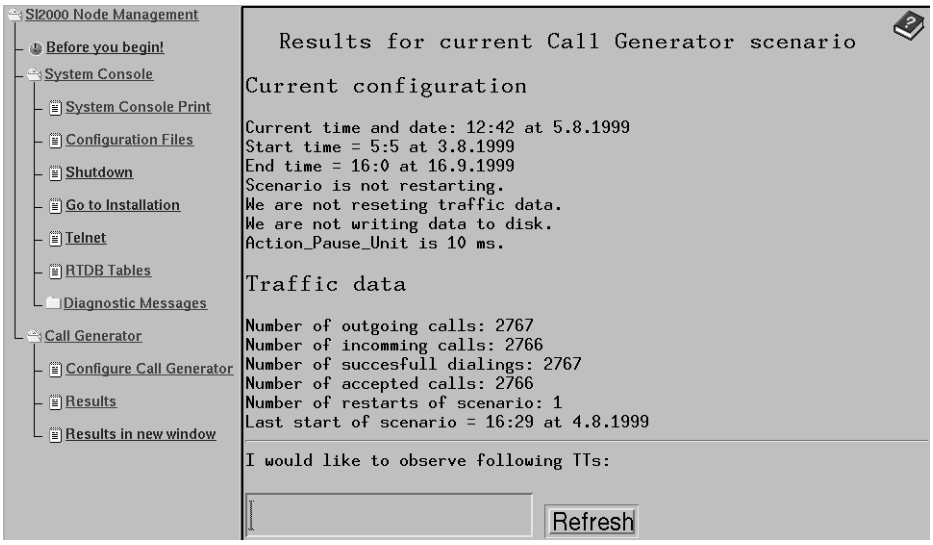


Figure 7: CG setup information and cumulative traffic results.

programs and included them as ADT (Abstract Data Type) [10] operators.

Conclusion

Basically there are two accessions to the CG use. The first one enables the user to program the CG to start its testing some time in the future. CG can be set to start the execution of scenario any date/time in the future. It can work, for example, from 10 pm to 5 am when most of the people are not working. A user can check test results when he/she comes to work. This kind of tests is preferred when user has to repeat the same scenario many times to verify the correctness and consistency of its code. But there is also a need for a quick test to be performed. This is usually very practical and useful in the earliest stages of the software development cycle. Quick changes in the database and online restart of CG through a favourite WWW browser covers this kind of tests. The first comments from CG users are very positive. They said that it is very practical and easier to learn compared to external testing equipment. They don't have to setup complicated configurations with a lot of external cables to connect the unit, since CG is part of SN. A developer does not have to leave its office, because all workstations and test SN are connected with Ethernet. That saves him/her a lot of time and inconveniences. Before CG implementation he/she had to go on the other side of the building, log to the local terminal, setup the SN and external test equipment and perform tests in noisy environment (SNs can be really noisy because of the large number of fans). We can assume that this approach will therefore stimulate developers to test their developing products more often. Hopefully testing in earliest stages will become the usual practice. If the result of CG use will be a better product and shorter development time, our goal will be achieved entirely.

References

[1] Software validation process.
URL: <http://www.bpr9000.com/validation.html/>.

[2] IskraTEL 1998. Iskratel annual report, 1998.

[3] IskraTEL 1997. *SI2000 Digital Switching System — User manual, System description.* IskraTEL d.o.o, Kranj, Slovenija.

[4] Mirko Šaranovič and Tomaž Mohorič. *Real time database (Only in Slovene).* IskraTEL d.o.o, Kranj, Slovenija.

[5] ITU-T Recommendation Z.120. CCITT Message Sequence Chart (MSC), 1996. Series Z: Programming Languages.

[6] Jan Ellsberger, Dieter Hogrefe, and Amardeo Sarma. *SDL Formal Object-oriented Language for Communicating Systems.* Prentice Hall Europe, 1997.

[7] ITU-T Recommendation Z.100. CCITT specification and description language (SDL), 1993. Series Z: Programming Languages.

[8] ITU-T Recommendation Z.100 Annexes C and D. Initial algebra model and SDL predefined data, 1993. Series Z: Programming Languages, Specification and Description Language (SDL).

[9] ITU-T Recommendation Z.100 Appendices I and II. SDL methodology guidelines, SDL bibliography, 1993. Series Z: Programming Languages, Specification and Description Language (SDL).

[10] Ferenc Belina, Dieter Hogrefe, and Amardeo Sarma. *SDL with Applications from Protocol Specification.* Prentice Hall International (UK) Ltd., 1991.