GENERATION OF REALISTIC SIGNALLING TRAFFIC IN AN ISDN LOAD TEST SYSTEM USING SDL USER MODELS

Thomas Steinert[1], Georg Rößler[2]

[1] *Univ. of Stuttgart, Institute of Communication Networks and Computer Engineering, Pfaffenwaldring 47, D-70569 Stuttgart, Germany, steinert@ind.uni-stuttgart.de*

[2] *Tenovis GmbH & Co. KG, Kleyerstraße 94, D-60326 Frankfurt am Main, Germany, Georg.Roessler@tenovis.com*

Abstract This paper describes the use of SDL for defining User Models for a test and measurement system whose purpose is to perform load tests of the control unit of an ISDN-PBX with realistic signalling traffic. The generation of realistic signalling traffic is based on the emulation of real users by User Models. The functional behaviour of a User Model is defined by an SDL process using a subset of the SDL syntax. The use of this type of User Models allows, besides the creation of messages according to distribution functions, to process messages received from the system under test and therefore to react appropriately on them, like a real user would do it. Normally several User Models are used together for a load test, so that the generated traffic is a realistic mixture of a variety of different user categories. The User Models are transformed to data structures for execution by the load test system. These data structures are optimized for efficiency in order to reach the target call rate of at least 30000 calls per hour on a standard workstation. The temporal behaviour of the User Models is defined separately by means of Event Generators which send signals to the User Models according to statistical distribution functions.

Keywords: Environment simulation, ISDN-PBX, performance testing, SDL, traffic generator

1. INTRODUCTION

Telephony is still the base function in today's PBX (Private Branch Exchange) systems, but customers expect much more than just telephone service. Some of these additional functions are implemented in the PBX as extensions of the switching software, like the support of mobile users with DECT (Digital Enhanced Cordless Telecommunications) technology. Other functions are realized by applications running on dedicated servers which are connected to the PBX. Call centers are the most prominent systems where server applications and the PBX together realize the desired functionality.

The main motivation to build the load test system is that the overload protection in the switching software has to be tested with realistic traffic. A second motivation for the load test system is that call centers require extensive testing under high and correlated load. Aspects that need to be tested are the interaction between the PBX and the servers as well as the correct handling of service requests by the call distribution according to the customer specific call flow scripts. Furthermore, the load test system allows to verify that a call center can handle the load which has been specified by a customer.

While there are quite a number of call generators commercially available, many of them are not able to generate the required call rate of at least 30000 calls per hour. Moreover, these call generators often lack some features which we think are important. The simulated behaviour must be the same as the behaviour of real users. This implies that all durations follow given distributions instead of generating calls with fixed interarrival times. Furthermore, the load test system must be able to react appropriately on the actions taken by the PBX, for example by repeating a call attempt after a call has been rejected. Finally, the load test system must be capable to produce load peaks like those arriving at call centers when a phone number is shown in a television commercial.

The focus for the load test system is on the switching software, whereas physical interfaces and the message forwarding by line cards can be tested using other tools. Hence the basic function is to generate and terminate calls at high rates with statistically correct behaviour in time. Statistical correctness means that the functional and temporal characteristics of the generated traffic are the same as from real users. An important feature is flexibility in the sense that it has to be easy to modify the simulated user behaviour and to introduce new signalling messages or even a new signalling protocol in the load test system. This flexibility has to be achieved without requiring significant changes to the load test system software.

This paper is structured as follows: Section 2 describes the developed method for the specification of realistic user behaviour by means of User Models where the definition of the functional behaviour is separated from the definition of the temporal behaviour. The architecture of the load test system and a basic test scenario are presented in section 3 and section 4 concludes the paper with some final remarks about the application of the load test system at Tenovis.

2. SPECIFICATION OF USER BEHAVIOUR

The main purpose of the load test system is to generate realistic signalling traffic, which allows to determine the behaviour of the system under test in heavy load situations. This leads to the question, how this realistic signalling traffic can be specified. The approach chosen for this load test system is to emulate each single user by a so called "User Model" which acts independently from the other emulated users. The resulting traffic of all User Models together represents a realistic signalling traffic load, as it would be created by a multitude of real users.

This section describes the developed method for specifying User Models emulating real user behaviour. Section 2.1 motivates the general principle for the method, 2.2 presents the description technique for the functional behaviour of a User Model which is based on the Specification and Description Language (SDL) defined by the ITU-T in [1] and 2.3 describes the way for specifying the temporal behaviour of a User Model, so that realistic traffic can be generated.

2.1. GENERAL PRINCIPLE

The generation of realistic signalling traffic of the load test system is, as described above, based on User Models. A User Model emulates the behaviour of a real user, i.e. it sends signals for triggering signalling scenarios and reacts appropriately on incoming signals depending on distribution functions.

To emulate the behaviour of a real user concerning signalling the scenario of a simple phone call is analysed. Its phases are sketched in Figure 1. There are two types of events occurring at the user during the phone call: "User Actions" representing actions triggered by the user himself and "User Receptions" caused by the PBX.

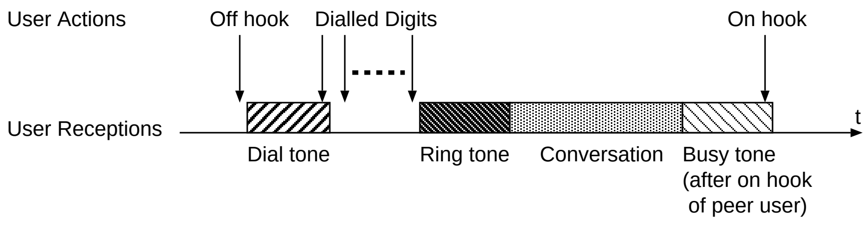

Figure 1 Phases of a phone call from user's point of view

Figure 2 contains a state-transition diagram of the FSM resulting from the scenario of the simple phone call. Again, there is the distinction between the "User Actions", which are sketched above the state symbols, and the “User Receptions”, which are sketched under the state symbols. The state-transition diagram uses the same “User Actions” as those of Figure 1, but different “User Receptions”. Instead of the continuous signals like e.g. “Dial tone”, atomic signals for the start of a tone and for the end of a tone are applied.

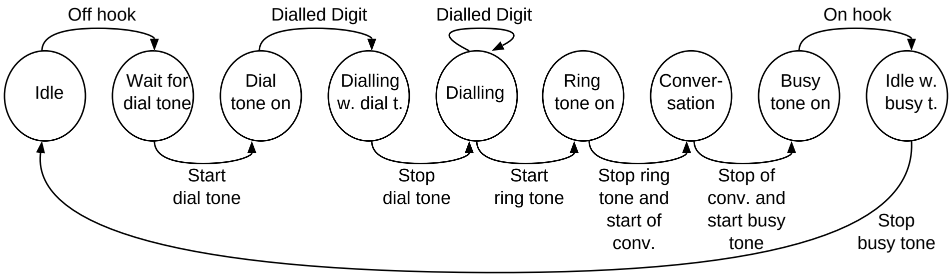

Figure 2 Modelling a user via a Finite State Machine (FSM)

It is important to understand that the “User Actions” are essential for generating realistic user behaviour, whereas the “User Receptions” are input signals from the environment. These “User Actions” require that the User Model is augmented by a specification of the temporal behaviour. The reason is that in order to obtain realistic signalling traffic, the time between the entry in a state and the arriving of a User Action has to be a random variable with a given distribution. The temporal behaviours of multiple User Model instances are independent of each other, although these instances may have the same statistical properties.

There are quite a number of publications ([2] provides a nice overview) about augmented FSMs which describe both functional and temporal behaviour. However, they mostly focus on the performance analysis and not on the emulation of behaviour in real-time. One exception is the paper by Lemppenau and Tran-Gia [3] which describes also a system for traffic generation.

Our approach is to separate the specifications of functional and temporal aspects as far as possible. The User Model FSM describes only the functional behaviour of users. The temporal behaviour is defined by means of so called “Event Generators”, which create signals called “Events” corresponding to the “User Actions”. These Events are one type of input signals of the User Model FSM. The principle of Event Generators is described in detail in section 2.3.

The second type of input of the User Model FSMs are “Signalling-SDUs” (Service Data Units) caused by the signalling messages sent by the PBX. These Signalling-SDUs (in the remainder of this paper referred to as SDUs) result from the layer 3 protocol of an ISDN signalling system. This protocol is realized as defined in the respective standard, but the interface to the upper layer is adapted to the needs of the load test system. Because several ISDN signalling systems, e.g. Q.931 [4] and 1TR6 [5], with different layer 3 protocols are supported by the PBX, the interface to the User Model FSMs has to be independent of the signalling protocol. Therefore protocol independent SDUs, whose definitions are adopted from Annex A of Q.931, are applied at the interface to the User Model FSMs. This type of signal is also used by the User Model FSMs to communicate in direction to layer 3 of an ISDN signalling system, so that the User Model FSM is able to control the underlying signalling protocol. In Figure 3 the interfaces of a User Model FSM are depicted: A User Model FSM receives Events from Event Generators to trigger actions, like setting up a phone call. It sends SDUs to the Signalling FSM for initiating signalling actions, e.g. for establishing a signalling connection to the PBX and it receives SDUs sent by the Signalling FSM indicating a state of a signalling connection.

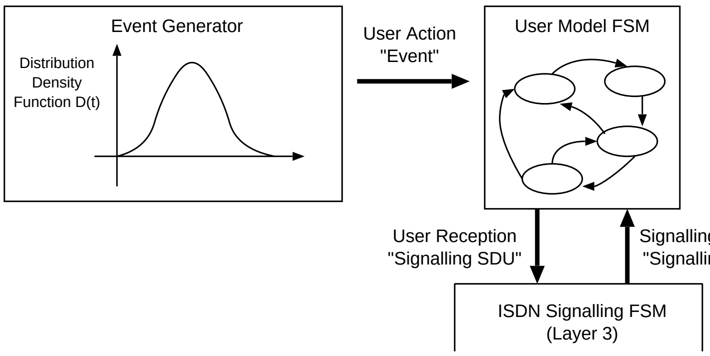

Figure 3 Input- and output signals of a User Model FSM

In the following section 2.2 the definition of User Model FSMs representing the functional behaviour of an emulated user by means of SDL is described.

2.2. SPECIFICATION OF THE FUNCTIONAL BEHAVIOUR USING SDL

Emulation of real user behaviour involves both the generation of signals according to a certain distribution and the reaction on the signals received from the PBX, depending on the current state of a signalling connection. Furthermore, the reaction could also depend on some special variables, e.g. the number of already repeated call attempts allowing to limit the number of repeated call attempts until a user would give up.

SDL is used to specify the User Model FSMs for several reasons. SDL is widely used in the software developing process in the telecommunication sector. Furthermore, commercial SDL tools can be used to create SDL specifications and to verify that their syntax and the static semantics are correct.

The load test system transforms the SDL specifications of User Model FSMs into data structures. During the execution of a load test these data structures are interpreted by the runtime system of the load test system. The data structures address implemented C procedures whose implementations are mostly short and simple. The main advantage of this approach in contrast to the generation of C code by the aforementioned SDL tools is the flexibility concerning the integration of new resp. the adaptation of existing User Model FSMs without a compilation process. Furthermore, the C procedures are part of the load test system and are therefore optimized concerning the performance, so that the real time requirements are fulfilled even if a large number of User Model FSM instances is active simultaneously. For complexity and particularly for performance reasons only a subset of the complete SDL grammar is used for specifying the User Model FSMs.

2.2.1 Applied SDL subset. For specifying the functional behaviour of a User Model the SDL subset contained in Figure 4 is applied.

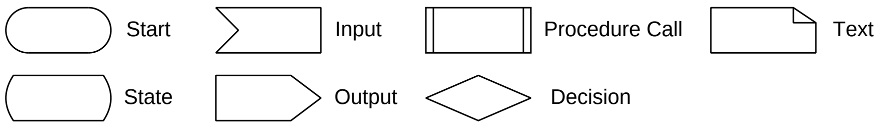

Figure 4 Applied SDL subset

The elements Start, State, Input, and Output are applied as usual in the context of User Model FSMs. The meaning of the other elements of the SDL subset is as follows:

Procedure Call: This element is applied for calling specific procedures. The indicated procedure name addresses a compiled procedure. There is a set of compiled procedures for the specification of the User Model FSMs, which can easily be expanded.

Decision: The decision statement allows to decide which branch of a transition has to be executed. The decision is performed by a procedure. The procedure name in this SDL statement addresses, as in the Procedure Call statement, a compiled procedure. Depending on the return value of this procedure, a branch of the transition is chosen for further execution.

Text: The text element is used for defining variables for the User Model FSM. Within the text symbol the variables are defined by applying the SDL DCL statement ("Declare"). Only the data types defined for the User Model FSMs can be utilized, but new data types can easily be added. The processing of variables of these types is performed in the compiled procedures which can be called by means of the Procedure Call statement. The utilization of variables within User Model FSMs is explained in detail in the following section.

2.2.2 Procedures and variables. The use of procedures and variables in the specification of a User Model FSM allows to introduce more sophisticated features. They permit to hide complex algorithms behind a simple SDL Procedure Call, e.g. for determining the current location of an emulated DECT user. Furthermore, it is possible to define application dependent data types, e.g. for the ISDN Signalling Information Elements, which are used within the ISDN signalling for negotiating the call parameters of an ISDN call.

To realize procedures and variables for User Model FSMs, there is a supporting library, the "FSM Runtime Library". This library contains the definitions of data types, which can be used within the SDL specification of User Model FSMs. Each data type of a User Model FSM has to correspond to a definition included in the FSM Runtime Library. Furthermore, the library contains the procedures applied in the SDL Procedure Call statements and the SDL Decision statements. The library is implemented in the C programming language and can be expanded by applying some simple interface rules. The relation between the SDL specification of a User Model FSM and the FSM Runtime Library is depicted in Figure 5.

A special data type is the one representing ISDN Signalling Information Elements. These Information Elements are used for specifying the parameters of an ISDN connection and for the application of supplementary services. They are realized within the User Models so that different services can be addressed without changing the implementation of the Signalling FSM.

2.2.3 Interfaces. This section describes the interfaces of a User Model FSM and particularly goes into detail to the relation between User

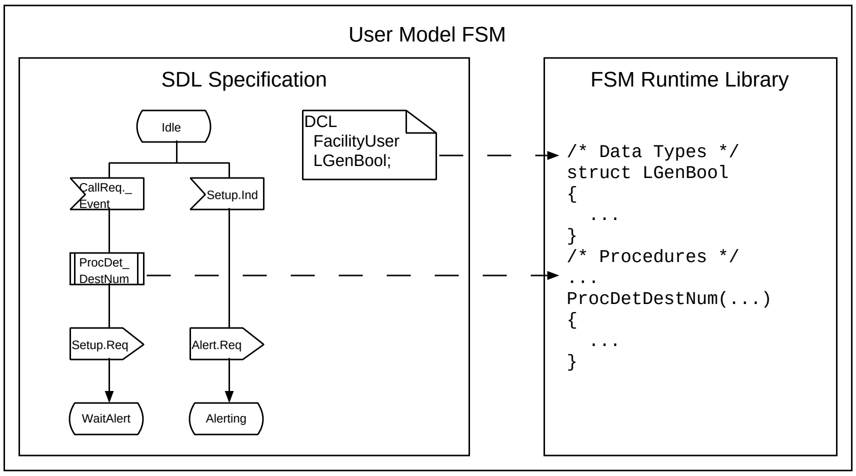

Figure 5 Relation between SDL specification and FSM Runtime Library

Model FSM and Signalling FSM. As described in the previous section, a User Model FSM disposes of interfaces based on messages to the Event Generators and to the Signalling FSM. The messages which are sent by the Event Generators are called "Events" and, therefore, are denoted in an SDL input statement with the suffix Event. The SDUs exchanged between User Model FSMs and Signalling FSMs follow the principle of the Abstract Service Primitives (ASP) defined in the ISO/OSI Basic Reference Model [6]: SDUs sent by the User Model FSMs to the Signalling FSMs have either the suffix Req or Resp, those sent by Signalling FSMs to User Model FSMs either Ind or Conf.

Figure 6 contains a signalling sequence diagram of a simple call with all signals exchanged between Event Generators and User Model FSM, User Model FSM and Signalling FSM and the resulting Signalling PDUs.

As it can be seen in Figure 6, a User Model FSM has a detailed view on the underlying Signalling FSM. However, the Signalling FSM contains all protocol internal procedures, like Timers or error recovery procedures. Therefore, and because of the protocol independent SDUs the User Model FSMs represent a high level of abstraction concerning the applied signalling protocols.

There are many SDUs sent by a Signalling FSM to a User Model FSM indicating a received layer 3 Signalling PDU which do not require a reaction of a User Model FSM, like MoreInfo.Ind in Figure 6. A User Model FSM can ignore these SDUs which results in simple User Model FSMs.

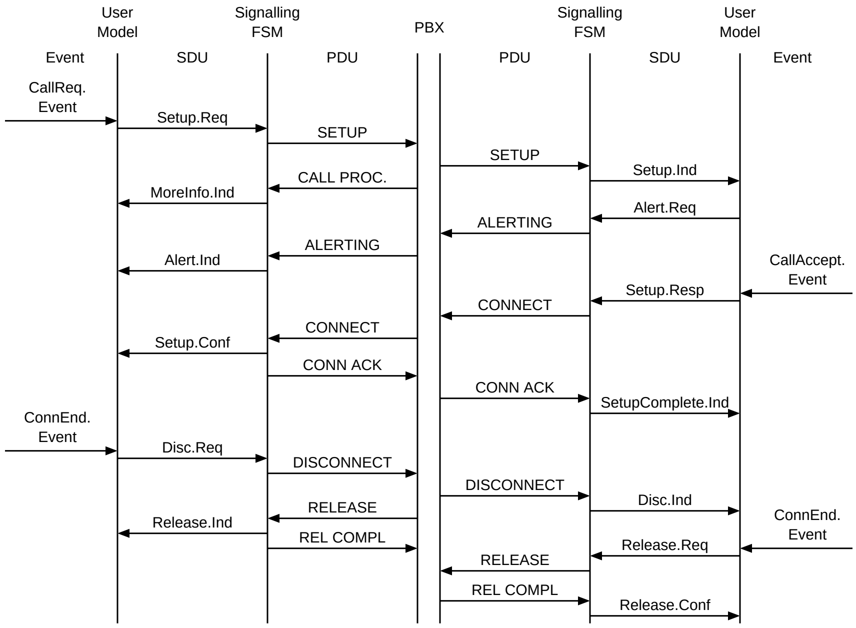

Figure 6 Complete signalling sequence diagram of a simple call

2.3. SPECIFICATION OF THE TEMPORAL BEHAVIOUR

As described in section 2.1, the specification of User Models for emulating real user behaviour is separated into two sections: The functional behaviour which indicates what an emulated user does, specified by means of an SDL process, and the temporal behaviour which defines when an action of a User Model has to be performed.

The realization of the method for specifying the temporal behaviour of a User Model is based on Event Generators. For each input signal of a User Model FSM of type Event there is an Event Generator defined. An Event Generator models a stochastic process by generating Events whose interarrival times are statistically distributed according to a defined distribution density function. The load test system disposes of an expandable set of distribution density functions, which are applied by the Event Generators for generating Events.

One purpose of the load test system is to generate statistically correct signalling traffic. If we analyse again the phases of a phone call illustrated in Figure 1, it can be seen, that only two types of Event Generators are sufficient to describe real user behaviour in an appropriate way: The "State Dependent" Event Generators which are related to a state of a User Model and the "Rate" Event Generators which are used for specifying the frequency of occurrence of an Event, independent from the state of the User Model. These types are presented in the following sections.

2.3.1 State Dependent Event Generator. Event Generators of the State Dependent type are related to the state of a User Model FSM, where the respective Event is specified as input signal. This means they are only active during the time the User Model FSM is in the respective state. Each time a state is entered, the respective Event Generators are started. This means, they determine the time when the Event should be delivered to the User Model FSM according to the configured distribution density function. If this time is reached, the Event will be delivered and the Event Generator will be stopped. If another input signal causes a transition of the User Model FSM, the Event Generator will be stopped before the Event will be delivered. An example for an Event generated by a State Dependent Event Generator in Figure 7 is the initiation of the "User Action" for accepting a call.

2.3.2 Rate Event Generator. In contrast to the State Dependent Event Generators are the Rate Event Generators not only active during a state of a User Model FSM. They are active as long as the corresponding User Model is active. This means they generate continuously Events, independently of the current state of the User Model FSM. This type of Event Generator is applied for specifying the call attempt rate of a User Model.

An Event of a Rate Event Generator is delivered to the User Model FSM at the time determined according to the configured distribution density function. But in contrast to the State Dependent Event Generator the time for the next Event is determined immediately after the delivery. If a User Model FSM cannot process an Event generated by a Rate Event Generator, because this Event is not specified as an input signal of the current state, the Event will be buffered until it can be processed. The motivation for this approach is that the load test system must be capable to drive the PBX into saturation. Some inaccuracies concerning the adjusted rate of Events can be tolerated. Normally, the configuration of the Event Generators of a User Model should be coordinated in that way, that no or only very short buffering is necessary.

3. LOAD TEST SYSTEM

The load test system "LGen" is implemented in software and runs on standard workstations. In the PBX, the real Interface Control Units

(ICUs) which terminate subscriber and trunk lines are replaced by a LAN ICU (LICU) which emulates Virtual ICUs (VICUs) on behalf of LGen. The messages from LGen are the same as those from real ICUs, thus the switching software running on the Generic Control Unit (GCU) cannot distinguish between real ICUs and VICUs. The LICU forwards these messages between the LAN and the PBX internal CBus.

The switching fabric is also involved in tests because the switching software requests B-channels to be through-connected.

3.1. ARCHITECTURE

This section presents the architecture of the LGen software, which is depicted in Figure 7. The architecture consists of the following building blocks:

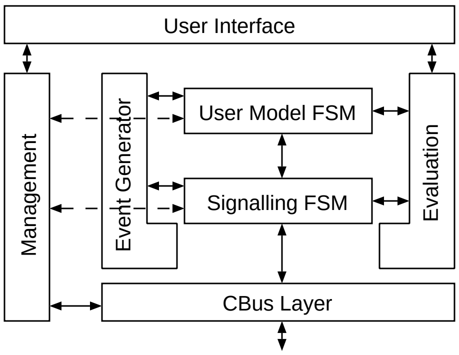

Figure 7 Software architecture of LGen

User Model FSM: This block contains an instance of a User Model FSM for each emulated user. The FSM Runtime Library is used for execution of the SDL processes. Each instance of a User Model is connected with an instance of a Signalling FSM. User Models communicate with these realizations of layer 3 protocols of ISDN signalling systems by means of protocol-independent SDUs. Within a load test multiple User Model FSMs can be applied simultaneously.

Signalling FSM: The block Signalling FSM contains the realizations of layer 3 protocols of different ISDN signalling systems. The messages exchanged between this block and the CBus Layer are Signalling PDUs, which are built according to the respective standard. The Signalling FSMs are implemented in the same way as the User Model FSMs, i.e. they are specified in SDL by applying the same method as described in section 2.2. LGen transforms these specifications into optimized data structures which are executed during a load test. Only the FSM Runtime Library differs slightly from the one of the User Model FSMs. This results in a higher flexibility concerning the supported signalling systems. As for the User Models it is also possible to run multiple Signalling FSMs with different layer 3 protocols simultaneously.

Event Generation: This block contains the Event Generators for the User Model FSMs generating Event messages according to defined distribution density functions. Furthermore, the Timers for the Signalling FSMs are realized within this block. To deliver the Event and Timer messages at the specified time, a Calendar module is applied. This Calendar is realized in a similar way than those for event-driven simulations, but in the case of LGen the messages are processed in real time.

Evaluation: This block contains modules for evaluating a load test. The required information is provided by the User Model FSMs and particularly by the Signalling FSMs. One type of information are the measurement values concerning signalling connections determined by the Signalling FSMs, like e.g. "Setup Answer Delay" or "Number of Accepted Setup". These values are passed to the Evaluation block, to determine results as e.g. mean value and standard deviation. Another type of information are the exchanged messages itself and the time when they were occurring. Therefore, SDUs and PDUs can be recorded during a load test in a file. Because of the large amount of recorded data it is possible to filter this message trace by user-selectable filters. This allows to examine in detail the exchanged messages or to determine the signal flow delay from one emulated user to another emulated user.

CBus Layer: The purpose of the CBus Layer is to put the received layer 3 PDUs in CBus messages and to send them through a TCP connection to the LICU. In the other direction it has to extract layer 3 PDUs from CBus messages received from the LICU. To perform these actions, the CBus Layer disposes of a table assigning CBus addresses of VICUs and ports to emulated users. A second task of the CBus Layer is to exchange the control messages between LGen and LICU for activating and deactivating VICUs and ports.

Management: The management block contains the modules for managing the entities of LGen. It administers the emulated users, i.e. it activates and deactivates the respective User Model and Signalling FSM instances, according to the user-defined test sequence configuration. Furthermore, it initiates the activation and deactivation of VICUs and controls the evaluation, e.g. by setting filters for the message trace.

User Interface: This block allows the interaction between the tester and LGen during a load test. A graphical user interface, which can be connected to this module, allows to initiate actions, like activating or deactivating emulated users, starting and stopping a load test or setting of filters for the message trace. Furthermore, it displays intermediate results of the evaluation of measurement values.

The communication between these blocks is mainly based on the exchange of LGen internal messages. These messages dispose of a fixed header containing information like sender and receiver address and message type. Depending on the type of the message there are supplementary message parts, like a complete Signalling PDU. This message based communication allows to decouple the execution of the different signal processing entities.

To take advantage of running LGen also on multiprocessor workstations, the developed program uses multiple threads to process the messages concurrently. The model applied is very similar to the "peer model" (also known as "workcrew mode"), where all existing threads work concurrently on their task, i.e. process LGen internal messages. This load sharing principle allows LGen to adapt to the underlying hardware platform in an optimized way. For a further description of thread based programming we refer to [7] and [8].

3.2. BASIC TEST SCENARIO

This section describes how the signalling traffic generation is realized with the architecture presented in the previous section. This is done by explaining how a load test of LGen takes place and how the blocks presented above interact during a load test.

After LGen is started, the Management is responsible for the control of a load test. It initiates the activation of the VICUs by sending a message to the CBus Layer. The CBus Layer exchanges then the control messages with the LICU through a TCP connection to activate the VICUs and ports. The activated ports are announced to the Management, so that only those users are activated, whose ports are already active.

The Management activates the users according to the test sequence configuration, which is provided at the configuration phase of LGen. To activate a user, the Management sends a message to the User Model FSM instance and to the respective Signalling FSM instance. The User Model

FSM instance starts the Rate Event Generators, performs the specified start transition and starts the State Dependent Event Generators of the first state. The Signalling FSM instance performs also the start transition.

If a User Model FSM instance receives an Event, it stops all active State Dependent Event Generators, performs the specified transition to the next state and starts then the respective State Dependent Event Generators. Normally, during a transition caused by an Event, an SDU is sent to the related Signalling FSM instance. Upon reception of an SDU the Signalling FSM instance performs the specified transition. If there is a PDU to send, the Signalling FSM instance builds the PDU, as defined in the respective standard, and sends it to the CBus Layer. The CBus Layer builds a CBus message containing the PDU and, among other information, the CBus address of the port, where the emulated user is connected. This CBus message is transmitted by means of TCP to the LICU, which sends the message via the CBus to the GCU, where it is processed. For the GCU there is no difference between CBus messages created by LGen and those of real ICUs.

The GCU transmits the messages for the ICUs (virtual and real ones) on the CBus. The LICU recognizes those messages which have to be forwarded and sends them through TCP to LGen. At LGen they arrive at the CBus Layer, which extracts the PDU, determines the emulated user by means of the CBus address and sends the PDU to the corresponding Signalling FSM instance. The Signalling FSM instance performs the transition which is specified for this PDU. If there is no transition specified, it will be ignored and the Signalling FSM instance remains in the current state. In most cases a PDU causes an SDU to be sent to the User Model FSM instance. The User Model FSM instance checks whether a transition is specified for this SDU. Only if there is a transition, the State Dependent Event Generators are stopped, the specified transition is performed and the State Dependent Event Generators of the new state are started.

To all SDUs and PDUs exchanged within LGen the filter rules of the message trace are applied. If a message has to be recorded, it is sent to the Evaluation, where the message and the current time are written to a file. The Signalling FSM instances perform the measurements during a signalling connection. At the end of the connection they send the measured values to the Evaluation, where the statistical evaluation takes place.

Besides the use of the test sequence configuration, it is also possible to control a load test by means of the User Interface. Upon user action the User Interface sends a message to the Management which interprets the message. It then determines the destination of the request and composes the appropriate internal message for e.g. the activation of emulated users, the setting of filters of the message trace or the stop of the load test. During the load test the Evaluation includes intermediate results in messages and sends them to the User Interface, where these results are displayed in an appropriate manner.

As described, LGen emulates real user behaviour as specified by means of User Models. As LGen is capable to process a large number of emulated users simultaneously with very low latency, the superposition of the signalling messages represents realistic traffic load. The concept of activating and deactivating emulated users is a natural way for emulating load peaks. It is sufficient to activate at a time several users, whose User Models are specified in that way that they start to initiate a call within a short time frame.

4. CONCLUSION

The LGen load test system is used mainly in the development departments and in the test floor. The first application was to verify that the existing overload protection mechanism worked as expected and to test an enhanced overload protection mechanism. In real systems, the observed fluctuations in call arrivals and other durations are due to the statistical nature of real traffic sources. LGen has demonstrated that it is important for meaningful load tests to use realistic traffic instead of using call generators which can only produce deterministic arrival patterns.

The target rate of 30000 calls per hour is clearly exceeded by LGen. The processor load is then still low which makes sure that the simulated behaviour conforms to the expected distributions. On a Sun Ultra 1 workstation, call rates of more than 100000 calls per hour can be reached. This very high call rate is useful for stress tests, even if the simulated timing behaviour deviates from the target distributions. This deviation is due to the non-negligible delays at LGen internal queues when the processor load becomes high.

A second important application area for LGen are call centers and other CTI (Computer Telephony Integration) applications. There are two main reasons why call centers have to be tested under high loads. First, some branches in the call flows which are executed by the external call distribution application are only used when the load is high. An example is a call flow which connects callers to an announcement for the estimated waiting time until an agent will answer the call. Another motivation for tests under high load is the experience that multiple systems from different vendors like in a call center do not always interact smoothly, especially if some systems operate close to their maximum load. LGen is used now in several areas in the context with call centers, and it has also been possible to demonstrate the performance of an operational call center at a customer site.

Work has started now to extend LGen to use it in the DECT context. LGen will be used to generate DECT-specific load peaks which are caused by high call rates and also high roaming and handover rates. A typical situation where such a scenario occurs is at the end of a big meeting when all participants leave the meeting room and many of them try to make calls.

LGen has proven to be a useful tool for tests during development and in the integration test floor. The key features are the realistic generated traffic and the flexible modelling of user behaviour. Both features result from the approach to model the behaviour in SDL and to use Event Generators for the execution in real-time.

References

[1] ITU-T Recommendation Z.100: SDL+ methodology: Use of MSC and SDL (with ASN.1)

[2] Mitschele-Thiel, Müller-Clostermann: Performance engineering of SDL/MSC systems, Computer Networks 31 (1999)

[3] Lemppenau, Tran-Gia: A Universal Environment Simulator for SPC Switching System Testing, Proceedings 11th International Teletraffic Congress (ITC), Kyoto 1985

[4] ITU-T Recommendation Q.931: ISDN user-network interface layer 3 specification for basic call control

[5] FTZ Richtlinie 1TR6: Kennzeichenaustausch zwischen DIVO(ISDN)-Vermittlungsstellen und ISDN-Teilnehmereinrichtungen – ISDN-D-Kanal-Protokoll

[6] ISO 7498, Basic Reference Model for Open Systems Interconnection

[7] IEEE 1003.1c-1995, Standard for Information Technology, Portable Operating System Interface (POSIX): System Application Program Interface (API) Amendment 2: Threads Extension (C Language)

[8] Nichols, Buttlar, Farrell: Pthreads Programming, O'Reilly&Associates, 1996