New Strategies for a Multi-System Simulator Supporting Complex Simulation Scenarios

Ulrich Fornefeld

Aachen University of Technology • Communication Networks • Prof. Dr.-Ing. Bernhard Walke
Kopernikusstr. 16 • D-52074 Aachen • Germany
Phone: +49-2 41-80 79 16 • Fax: +49-2 41-88 88-2 42
E-mail: ulrich@fornefeld.de • WWW: http://www.fornefeld.de

Abstract — **This paper provides concepts for the performance evaluation of wireless telecommunication protocols using complex simulation scenarios for a most realistic interference situation. As both tasks, the simulation of the protocol stack as well as the calculation of the interference calculation are quite resource-consuming for complex scenarios, special care has to be taken not to exhaust the hardware limitations of the simulating machines.**

I. Introduction

Many present simulator concepts aim at either the implementation of protocol stacks or the simulation of the interference situation and it's influence on the radio resource control (RRC) algorithms. As these two concepts target different simulator architectures, which are both very time-consuming, a combination of both has not been possible in the past due to lack of calculation power.

A lot of simulations of either the complex protocol stack with few participants or the comparably simple radio resource simulation with many instances have been performed. there was, however, no exact way to examine the dependencies between the interference situation and the performance parameters of the protocol stack.

Nowadays the required hardware resources are present for a combined strategy, so an approach shall be made to integrate both concepts into one simulator.

This system, the *SDL-Generic Object-Oriented Simulation Environment* (S-GOOSE, Fig. 1), aims at the investigation of protocol improvements for mobile communication systems. Knowing the exact interference situation, flow control algorithms might be added to the system adapting the behaviour of the protocol stack to the present interference situation.

In Sec. II, the basic concepts of the S-GOOSE are introduced. Sec. III and Sec. IV explain the details of the modelling concepts for the scenario and the protocol stack, Sec. V explaines some helpful features for huge simulation runs. The present state is described in Sec. VI

II. General Modelling Concepts

S-GOOSE provides an object-oriented multi-system simulation environment, containing a defined interface to the *Specification and Description Language* (SDL). It is based on the C++ class library *System PErformance Evaluation Tool Class Library* (SPEETCL),[1].

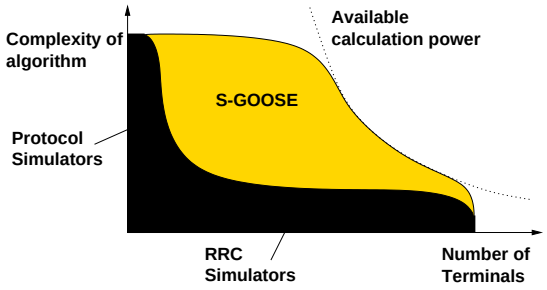

Figure 1: Common protocol- and RRC-simulators vs. S-GOOSE

Time consuming algorithms like C/I calculation are implemented in C++ whereas algorithms demanding a high flexibility and intuitive understanding are specified in SDL. As both, the C++ language as well as SDL, are well-known in the telecommunication area, a high acceptance level of the new structure is expected.

The specifications designed in SDL are translated to SPEETCL classes using the code generator SDL2SPEETCL [2]. The generated C++ classes will be linked together with the classes manually written in C++. The non-generated classes form, in terms of SDL, the environment of the systems which can be reached via signals.

There is no restriction to a single telecommunication system. Multiple systems might be instanciated at runtime to perform coexistence investigations. The necessary system-dependent algorithms are added by loadable modules for a high flexibility, Fig. 2. When a module is not needed, it is not loaded and therefore memory resources are saved.

Many simulators are based on the two concepts mentioned above, so it is possible to establish a fixed central part for future simulators, that can be reused and that is common to all simulators. This part is completed by user-defined modules. In this way, man-power is saved using this concept.

III. Concepts for Scenario Modelling

The kind of choosen scenario has a remarkable influence on the accuracy of the obtained results as well as on the runtime of the simulation. Therefore, the scenario model has to be highly adaptable, which demands a large amount of modules and it has to be implemented run-time optimized.

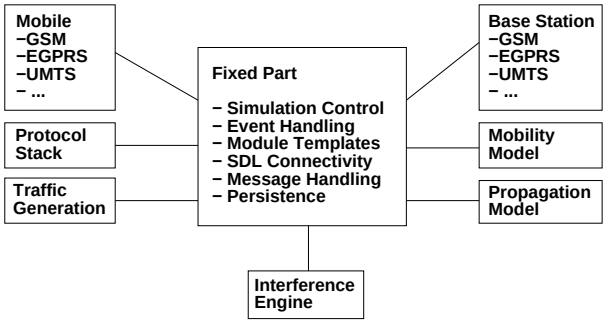

Figure 2: S-GOOSE: Fixed part and selectable modules

This is achieved by implementing the scenario in C++classes manually.

A detailed channel model is used to find out the current interference situation. A variety of different interference calculation methods allows the selection of a performance-optimized or a more accurate simulation run considering aspects like co channel interference and adjacent channel interference, background noise or (for multi-system examinations) general interference models. An interferer administration is not applied as this would result in an increased run time when frequency hopping is applied.

The propagation model might be selected from different analytical models (e.g. Hata-Okumura) or might be obtained from a map created by a frequency planning tool. Different shadowing and fading effects might be added, the influence of several antenna types, directed and omnidirectional, might be observed as well. For signal quality measurements, a mapping from the C/I ratio to bit errors is applied. Different mapping methods like polynom calculation, look-up from a table or stochastic methods are available.

IV. Concepts for Protocol Modelling

The Specification and Description Language (SDL) has become a powerful instrument for protocol and algorithm specification for telecommunication developments. Present ETSI standards are providing algorithms and algorithm parts as SDL code, e.g. HIPERLAN/2. To support the SDL language, an interface for the usage of SDL specifications is provided. The specified algorithms are highly comprehensive because of their graphical representation. They can be adapted very easily without getting in contact with a textual programming language.

For a most flexible architecture, only the skeleton of the OSI stack is implemented fixed and the selected algorithms are loaded at runtime into the generic protocol stack, Fig. 3. The layer manager generates the module instances when the instanciated system demands the module to exist in the specific layer. The module entity loads the demanded algorithm type. In this way, a module is not fixed to a layer. This is necessary as, e.g. the mobility management is not located within the same layer in all standards. In addition, once a layer is selected for a module, every available algorithm might be loaded and a reuse is possible as well.

As an example, Fig. 4 shows the loadable algorithms for the power control. Management instances will route the incoming signals to the existing algorithm instances. In this way, the management instances assert the existence of the algorithms, the environment of the algorithm is independent of the choosen algorithm type.

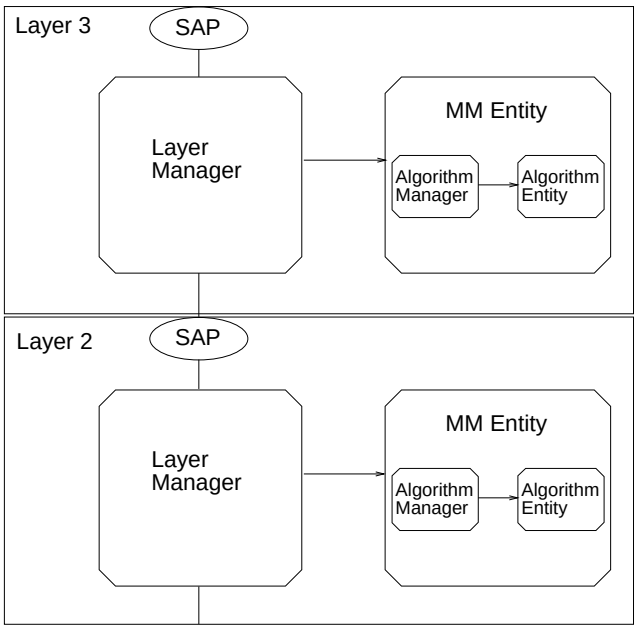

Figure 3: Generic layer structure

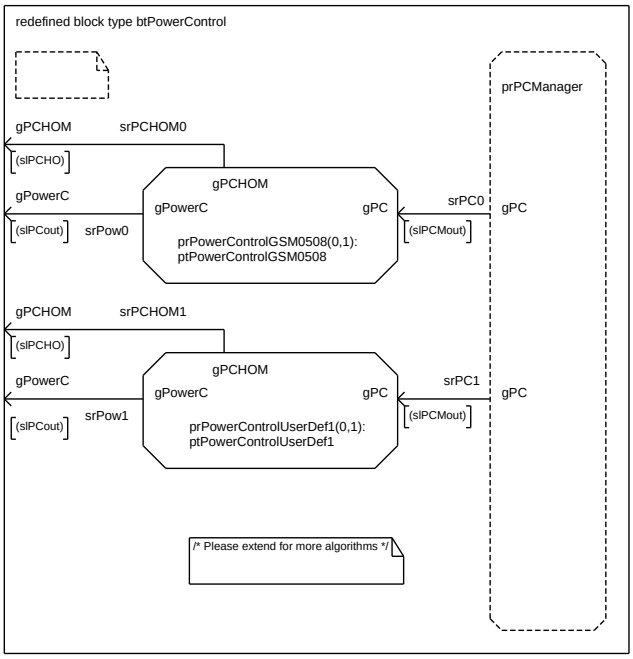

Figure 4: Loadable algorithms for the radio resource management, Power Control

The provided SDL code might be adapted by the user, further algorithm types might be added. The compilation to SPEETCL-Classes using the SDL2SPEETCL compiler allows easy linking to the simulator kernel.

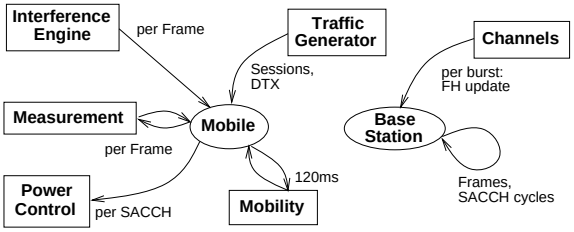

Figure 5: Frequent events in a GSM system

V. Helpful Features

A. Event-Driven Simulation

The simulation time advance is organized by sending events to an event scheduler. This central module of the simulator manages the incoming events. Each event is send to a certain time in future provided during the event generation. When the simulation time reaches this arrival time, the scheduler will send the event to the receiving module, the event arrives. Each module is able to receive certain events and to handle them. As SPEETCL is a pure event-driven tool, there are no processes defined. Processes specified in SDL will be translated to events and methods by SDL2SPEETCL. The clearness of this concept - providing only events for the time advance - is not only very comprehensive but also allows a simple implementation in C++.

Event handling The basic stucture in Fig. 5 displays the cyclic events for the GSM system. They form the functional skeleton of the simulator, advancing bursts, frames and other regular events. Each module manages it's own regular events, mostly in an abstract base class, so there is hardly any coordination necessary between the modules. This simplifies the support of different systems.

If selected, the S-GOOSE modules will generate a message for each received event containing a time stamp and the source location. This option can be selected per module.

B. Persistency

For rare event simulations beeing quite time consuming, a method of repetition of rare events, called RESTART algorithm (REpetitive Simulation After Reaching Thresholds, [3]), has been investigated. This method shortens the rare event simulation remarkable. When observing a variable with rare event character, a threshold for a seldom but not rare state will be defined. A simulation run determines the probability of the seldom state. In addition, whenever the seldom state is reached, the state of the simulator is saved. From the saved states, a state is randomly selected and the simulation is restarted with a new, randomly determined seed for the central random number generator. In this way, the probability of reaching the rare state from the seldom state can be determined. The absolute probability of the rare state is obtained by multiplying the two probabilities determined above. This algorithm might be extended to multiple thresholds. The optimum number of threshold levels can be calculated. For this approach, a feature called persistency is necessary, allowing to save all internal states of the simulator to a file and to restore the state in a further simulation run. Every object provides a method for the storage process and one for the restore process, besides it has to be marked with a unique object id to support the restore process. When the save process is initiated, the persistency control will save the objects existing at this simulation time. When restarted, the restore process will reate all objects and initialise them with the saved states.

As a side effect, persistency allows to handle with a bundle of problems that might interrupt a long simulation run, e.g. lack of memory, timeouts of the queueing systems, bugs, hardware problems or power cuts. The persistency prevents from a loss of a long simulation run. In case of an interruption, the simulation can be restarted at the last saved state when frequent saves were performed.

C. Evaluation Methods

SDL is a specification language, that does not provide methods for performance evaluation such as recording and evaluation of stochastical data. To avoid this obstacle, additional features have been introduced to SDL. As the additions are not standardised yet, they are defined using comment symbols in the specification. The SDL2SPEETCL compiler will understand the MSG directive for the creation of debug output messages and the PROBE directive for a probe concept collecting data and doing statistical evaluation.

The possibility of writing probes from the SDL specification allows a comfortable way of performance analysis. Probes might record either values, collected at one SDL symbol of the specification, or time advances between two different SDL symbols. Therefore different probe types are defined that can be evaluated in several ways for the best results presentation. Simple evaluation methods like a calculation of the moments might be used as well as sophisticated methods like a probability density function, asserted with a Batch Means algorithm or a Limited Relative Error (LRE, [4]) algorithm. Messages can be sent as debug information using a predefined message channel. In addition, a message sequence chart (MSC) can be generated. A MSC shows the sequence of signals, states and actions and makes the specification very comprehensive.

VI. Conclusions

In this paper a multi-system simulation environment has been developed in order to allow extended protocol investigations. A detailed scenario model will provide the necessary information about the interference situation. The capabilities of the protocol to handle with the interference is observed within the generic protocol stack.

A GSM system has been implemented to proof functionality, already providing the basis for the GPRS protocol stack that will follow soon. The attemp to create a generic protocol stack has shown many overlaying functionality common to several wireless systems, that is reused within the simulation environment.

SGOOSE and SPEETCL are available at AIXCOM Gesellschaft für Telekommunikations-Dienstleistungen mbH (www.aixcom.com) which is a spin-off of the chair of communication networks (ComNets). Both have been tested with Solaris 2.6, 2.7 and Linux 2.2.16 using the gcc compilers 2.8.1 and 2.95.2.

VII. References

[1] M. Steppler, *SPEETCL — SDL Performance Evaluation Tool Class Library*. AixCom GmbH (www.aixcom.com), Apr. 2000.

[2] M. Steppler, *SDL2SPEETCL — An SDL to C++ code generator*. AixCom GmbH (www.aixcom.com), Apr. 2000.

[3] F. Schreiber and C. Görg, "Rare event simulation: a modified restart-method using the lre-algorithm," *accepted contribution to the 14th Int. Teletraffic Congress*, June 1994.

[4] F. Schreiber and C. Görg, "Stochastic simulation: a simplified lre-algorithm for discrete random sequences.," *AEÜ, 50*, pp. 233–239, 1996.