# SIMULATION CATEGORY LANGUAGES - A DDP EXAMPLE

P. Nick Lawrence
Advanced Software Technology
Texas Instruments, Inc.
Austin, Texas

## ABSTRACT

General simulation languages can be used to express any simulation situation. Often, however, the expressions are bulky and cumbersome because of the host language's need for generality. High-level languages for specific situations or problem areas, called "Category Languages", can greatly improve users' access to simulation computing power and can quite easily and justifiably be implemented as "front-end" pre-processors to ordinary general simulation languages.

In this paper, definitions and motivations for Category Language Methodology are presented, with a discussion of applications areas, implementation methods, and an example of the Category Language Methodology drawn from Distributed Data Processing.

## CATEGORY LANGUAGES

A Category Language is a programming language tailored to a specific, limited-context applications area. Category Languages will normally be defined and implemented to solve isolated problems in a convenient fashion. For example, one builds a Category Language whenever a set of utility subroutines are written to support a programming task. The subroutines make it convenient to handle similar program situations. They are normally not of general utility.

By giving up generality, one may greatly simplify and shorten investigation time for constructs in the applications area. Common tasks and activities are accomplished in a type of short-hand, increasing efficiency and reducing the likelyhood of error.

Simulation languages exhibit several alternate world-views (cf. 1). These world-views are necessarily broad perspectives of the simulation arena. The ability to select more limited arenas may enable much more powerful constructs to be developed.

Investigation of Category Languages as a simulation methodology may lead to improved capabilities in general simulation languages. Currently available languages are really low-and medium- level languages. One has not yet appeared which does for simulation what (for example) PASCAL has done for computation. Category Languages will help to more fully explore the possibilities for "structured simulation". Such experience must be valuable in the effort to advance simulation methodology and produce more powerful tools.

## FEATURES OF CATEGORY LANGUAGES

A Category Language allows the expression of major constructs in simple terms. Large pieces of a problem area can be represented in single, parameterized structures in many cases. Expressing a particular problem in such structures is often faster, easier to understand,and less error-prone than re-programming each new case.
The limited context limits the proliferation of such constructs, making Category Language design and implementation feasible. If the context is not limited, as in current simulation languages, a large number of major capabilities are necessary. Construction of a special-purpose language, a Category Language, can be the most efficient and cost-effective way to approach the problem.

A Category Language can be tailored to meet the needs of a specific project and a wide range of users, while not limiting the project's simulation specialists. A primary use of a Category Language is to remove the simulation specialist from the construction and operation of routine project simulation tasks, freeing him to concentrate on exceptional cases. The general project staff is thus provided a tool to examine most of their problems and a relatively unloaded consultant, the simulation specialist, to handle the exceptions. The specialist can build from the Category Language to handle special cases, concentrating on the extensions and not the body of the problem.

Use of Category Languages can improve engineering and managerial efficiency and cost-effectiveness in many areas. The many reasons for using simulation have been documented elsewhere (cf. 2). Category Languages exhibit these benefits and add others. Models can be varied by people who don't know the workings of a simulation system. Model construction is usually faster and verification easier due to smaller source modules. Users can concentrate more on the problems and less on the tools. All of these can result in cost savings and efficiency.

## TYPICAL CATEGORIES

There are as many categories as there are points of view about problems. Applications areas can be conveniently divided into the management-oriented categories and the problem-oriented categories. The difference lies more in the point of view than in the functional details. Management-oriented categories are essentially a matter of the amount and type of detail, while problem-oriented categories deal with specific problem areas.

The two types are really orthogonal. Given a problem area, the level of detail is usually a design option. (The problem area may need to have a certain level of detail, but that doesn't guarantee that that level will be implemented.) And certainly the decision to have a specified degree of accuracy in the model does not dictate what problem area the model will address.

Two convenient management-oriented categories are proposal-level simulation and project-level simulation. A proposal-level simulation investigates a variety of system alternatives, while a project-level simulation fine-tunes a specific system.

In a proposal situation, one needs insight and reassurance and has very little time to gain it. Category Language methods can provide what is needed on time, by letting the engineer investigate many more situations than would otherwise be possible.

In a project situation, one needs detailed, on-going support of a specific design in order to optimize and perform trade-off analyses. Category Languages can remove the constant need for simulation consulting during the project by giving engineers and managers easy access to a simulator and freeing the consultants for special efforts not efficiently covered by the Category Language.

There are many problem-oriented categories. The primary features of a problem-oriented Category Language are a few high-level structures which represent large pieces of a problem area. This coverage is illustrated in the next section.

## A DDP CATEGORY LANGUAGE

An example of a problem category for which a Category Language has been developed and implemented is a type of Distributed Data Processing (DDP). A system composed of many identical but independent processors configured in some topologic network is a common engineering problem. One is interested in relative loading and capacities, bottlenecks and under-utilization. A Category Language (NETSIM) has been designed by the author to explore such systems.

NETSIM is a problem-oriented Category Language aimed at proposal-level studies of identical, independent-node DDP systems. A stochastic approach is assumed. An inventory of various kinds of nodes, lines and tasks is provided. Nodes and lines are slaves to tasks, and a description of the network and tasks describes the system.

NETSIM will be described to illustrate a problem-oriented, proposal-level category language and not to document NETSIM. The reader can easily build his own version of this Category Language if he is interested.

Keywords are provided to define broad capabilities of nodes. Examples are SENDER, RECEIVER, SHOTGUN, COUPLER, and REACTOR. The keywords selected closely suggest the node function. A SENDER originates messages, i.e. a message source; a RECEIVER is a message sink; a SHOTGUN sends on all of its outgoing lines at once; a COUPLER relays messages; a REACTOR responds with a return message. There are other types, but these illustrate the point.

Some combinations are allowed, such as SHOTGUN COUPLER.

In the NETSIM language, one can define various node parameters such as clock times, traffic, etc. or take the defaults. A node is described and given a variable name.

Nodes are connected pairwise by lines. Lines are passive and controlled by nodes. They are described as bandwidths and endpoints. They are named and typed either by default or explicit definition. The network digraph structure is then described by using named nodes and named lines.

For example,

```
SENDER NODE ALPHA.
RECEIVER NODE BETA.
LINE GAMMA.
GAMMA FROM ALPHA TO BETA.
```

Here nodes ALPHA and BETA are typed, receiving several default parameters by omitting specific overrides. Line GAMMA is similarly specified. The topology of the network is clear from the final command. (See fig. 1).

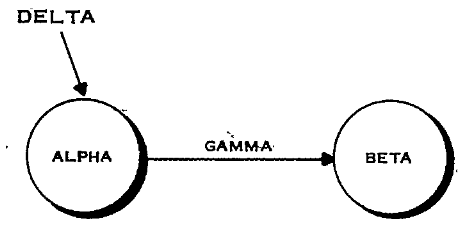

Figure 1. A Simple NETSIM System

The load for a system may be specified many ways. One is to define a construct called a task, which directs node activity. Tasks in NETSIM are composed of segments of node-use followed by I/O on selected lines, described statistically (fig. 2). Typical parameters are the number of segments, the probability of an output after each segment, the number of node clocks per segment, and the size of an outgoing message.

For example,

```
TASK DELTA, 50 SEGMENTS.
LOAD DELTA INTO ALPHA.
```

This defines task DELTA by accepting the defaults (except the number of task segments, here specified as 50), and establishes work for the system by assigning the task to node ALPHA.

Similar methods are used to specify what to measure, what to print, how long to run, etc.

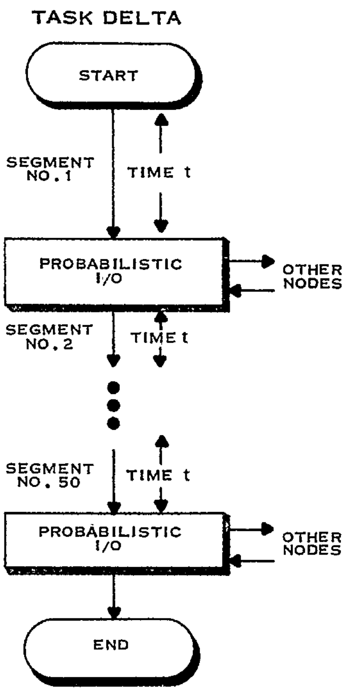

Figure 2. Task Delta

It is clear that very little knowledge of simulation techniques would be needed for a user to build a large simulation in NETSIM. Of course, a knowledge of good experimental techniques is still required.

## METHODS FOR IMPLEMENTING CATEGORY LANGUAGES

The chosen Category Language must easily represent large pieces of the problem area. It should probably be designed context-free for easy implementation. A trade-off must always be made between Category Language implementation effort and the savings from Category Language use.

The basic elements of any computer language compiler are "scan", the parsing of the source, and "code gen", the construction of an executable representation of the source.

To minimize time and effort, the scan portion should be built in a very modular and straight-forward manner. The author has successfully used SNOBOL as the host scan language because of the ease of describing and discovering syntax in a source code being scanned. Automatic compiler-writers exist and might prove more suitable (cf. 3). Simplicity is of paramount importance.

In NETSIM, the target language is GPSSV. Source is translated into GPSSV as macros, "canned code", and a variable data base. The dynamic code generation of standard compilers has been studiously avoided.

Parameterized GPSSV simulations were built for each basic category structure, i.e. node types, lines, etc. These were debugged by direct execution. All possible data was carried in matrix savevalues, thus avoiding GPSSV code structure variations. The NETSIM compiler simply fills the matrix savevalues and transcribes the necessary reentrant code blocks. Execution of the resulting GPSSV program is simply a matter of supplying it as source to a GPSSV system.

NETSIM was built in one man-month and is about six hundred source lines of SNOBOL.

It can be seen that these methods will produce effective Category Languages for suprisingly little person and computer time. Important areas which must be addressed are compile-time diagnostics, modeling and experimental techniques, and of course, validation and verification. A cost-benefit trade-off must be done regarding diagnostics. As for the last two, Category Language simulation is intended to minimize simulation specialist involvement, not necessarily eliminate it. A simulation consultant should be used by a project requiring extensive simulation.

Verification and validation are always thorny issues. However, the ease of use of the Category Language encourages many runs which can be mutually cross-checked. Insight is always the ultimate driver of verification and validation.

## CONCLUSIONS

The Category Language offers the user quicker, simpler and more ready access to a relevant simulation. It can be implemented cheaply, quickly and effectively. For many projects, it is a viable alternative.

The Category Language may promote the development of simulation toward more powerful and general simulation methodologies.

## BIBLIOGRAPHY

(1) I. M. Kay, "Digital Discrete Simulation Languages: A Discussion and Inventory," *Fifth Annual Simulation Symposium*, Annual Simulation Symposium, Tampa, Florida, 1972.

(2) T. H. Naylor, *Computer Simulation Experiments with Models of Economic Systems*, John Wiley and Sons, New York, 1971.

(3) David Gries, *Compiler Construction for Digital Computers*, John Wiley and Sons, New York, 1971, esp. chap 20.