# GENERATING SYNTHETIC WORKLOADS FOR REAL-TIME SYSTEMS

**D. L. Kiskis and K. G. Shin**

*Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122, USA*

Abstract. In this paper, we describe a software system which generates synthetic workloads for use in the performance evaluation of distributed real-time computer systems. The software system consists of a high-level description language and its compiler. The language provides a flexible, easy-to-use description of the structure and behavior of the real-time workload. The compiler, called a synthetic workload generator (SWG), uses this description to produce an executable synthetic workload (SW). The SW may then be used to drive the system under evaluation while measurements are being made.

Keywords. Computer selection and evaluation; real-time computer systems; software tools; specification languages.

## INTRODUCTION

Real-time systems have strict performance requirements. To determine if these requirements are met, the performance of a system is evaluated through experimentation. During the experiments, the values of selected performance indices are measured while the system is running a workload. The selection of the drive workload directly influences the results of the evaluation.

One possibility in the selection of the drive workload is to use the actual application software. However, there are a number of situations where the real workload is unavailable or unrealistic. Such situations include new systems where an application workload has not yet been developed and critical systems where, for safety reasons, performance evaluations must be done off-line. In these cases, we advocate the use of an SW as the drive workload. An SW consists primarily of a set of parameterized synthetic application tasks (SATs) which execute on a system and produce demands for resources. It also includes a driver task which controls the actions of the SW to facilitate the use of the SW during experimentation. It controls when the SW starts and stops. It also determines when the individual tasks execute.

In this paper, we describe a suite of software tools which we have designed and implemented to support the specification, generation, and execution of SWs for a distributed real-time system. This suite provides the high level support necessary to efficiently produce SWs which are customized for a particular evaluation. The suite consists of the synthetic workload generator (SWG) and some minor support programs. The SWG compiles a description of the workload that is specified in the synthetic workload specification language (SWSL). SWSL describes the structure of the SW based on a dataflow model.

The work reported in this report was supported in part by the NASA under Grant No. NAG-1-296 and NAG-1-492 and the Office of Naval Research under Contract No. N00014-85-K-0122.

There are two primary goals in the design of the SWG suite. The first is to be capable of accurately representing actual real-time workloads. This goal is met through the selection of an appropriate workload model. The model was chosen to reflect the structure of the software which composes the workload being modeled. By accurately modeling the structure of the workload, we also capture many of its behavioral characteristics. Representativeness is enhanced by the selection of parameters for the objects in the workload. Parameters are defined for both the SATs and the resources that they use and, possibly, share. These parameters were selected to reflect both common software properties and those properties which are specific to real-time software.

The second goal in the design of the SWG suite is ease of use. All components of the suite should be easy to use while retaining their flexibility and power. Ease of use is enhanced by the simple, regular structure of SWSL. The language structures allow one to change both the values of parameters and the interactions between SATs with little effort. We also provide a simple user interface to the SWG. It is completely automated to handle all the various compilation stages and their corresponding intermediate files.

This paper is organized as follows. In Section we describe the notation used to specify the SW. In Section we discuss the functions of the SWG, and in Section we discuss the SW which is being supported by the SWG suite. Section we give our summary and discuss our future work.

## THE WORKLOAD MODEL

The workload model provides a high level description of the structure of the workload. We represent this structure using a dataflow notation. A dataflow notation was chosen because it is commonly used to specify software structure. Workload specifications in other dataflow notations may be

easily translated into our dataflow notation. The translated workload will retain the structure of the original. Hence, it will be quite representative. Using our notation, we can specify both the individual tasks and the interactions between tasks.

Task Level Notation

The notation is divided into two levels of abstraction. The higher level, or *task level*, defines the tasks, the resources they use, and their interactions. The task level notation uses formalisms borrowed from the area of structured analysis (SA). In particular, we base the notation on ESML, an SA notation created by Bruyn and others (1988). ESML was developed for the high-level specification of real-time software. It is a combination of the Ward/Mellor (Ward, 86; Ward and Mellor, 1986) and Boeing/Hatley (Hatley and Pribhai, 1987) SA notations. These two notations were independently derived and use differing approaches to add timing and control information to the basic data flow model developed by DeMarco (1978).

By basing our notation on the SA notation, we accomplish our primary goals. First, we tie the structure of the SW directly to the structure of the workload, thus improving the ability of the notation to accurately model actual workloads. Our notation is the first to be based on a high-level software specification notation. Previous systems were based either on low-level specifications such as flowcharts (Baird, 1973; Walters, 1976) or on high-level notations such as UCLA graphs (Singh, 1981) which are not related to software specification notations. Second, we make it easier for the user of the SWG suite to produce workloads. The SA notations are commonly used by CASE tools for high-level software specification. Hence, it is likely that the actual or proposed workload being modeled has been specified in terms of an SA or similar notation. To produce a description of the workload in our notation, the user must translate the specifications. This process may be performed manually, or may be automated as part of a CASE tool. By using a similar notation, we simplify the translation.

SWSL defines the workload in terms of transformations, flows, stores, and terminators. Transformations represent units of computation, generally tasks. Flows are data and control paths. Stores are units of data storage, and terminators are interfaces between the workload and the environment. The parameters for these objects define characteristics such as task interactions, scheduling requirements for tasks, and access properties of shared objects.

Operation Level Notation

The lower level abstraction in the notation is the *operation level*. It defines the task's internal structure, behavior, and the manner in which it uses resources. This notation is similar to that used by Singh and Segall (1982) in the Pegasus system. A task is defined in terms of sequences of operations and control logic. Each operation represents the use of a single resource by the task, and the control logic determines the sequence in which the resources are used.

The control structures consist of loops and branches. They execute probabilistically to simulate the variation of program execution based on the value of the task's input data. Hence, the SATs simulate the random execution time distributions of real application tasks. The control structures also cause the SATs to simulate the resource usage patterns of the real application tasks and not just the quantities of resources used. By simulating the random execution times and the resource usage patterns, the SW models the workload more realistically. This realism is necessary when studying real-time systems. The SW must express the time-specific behavior of the workload. It is this behavior which affects the real-time aspects of the system.

## THE SYNTHETIC WORKLOAD GENERATOR

The SWG compiles the SWSL specification to produce the SW. It reads the task level description and produces parameter tables. These tables describe the structure and parameters of the task level notation in a form that may be used by the SW driver. The SWG compiles the operation level description to produce C code. Each operation in the description is expanded into its equivalent code. This code is stored in a library containing code for all possible operations. Later, the SWG invokes the C compiler to create the object code for the SATs. This object code is then linked with the parameter tables and the object code for the SW driver to produce the complete executable SW.

The SWG offers a number of support features to aid in the creation of SWs. It performs syntax and semantic error handling on the input files. It also does consistency checking on the dataflow graph for the workload. It enforces the construction rules for the notation, thus reducing the probability of logical errors in the SW.

The SWG provides another important feature. It supports the automatic creation of replicated objects from templates in the SWSL specification. This feature is used when multiple tasks in the workload have the same parameter values. The user specifies the structure of one instance of the task. The task definition states that a copy of the task be executed on each of a number of different processors. Those copies are then generated automatically. The user does not need to individually program the specifications of each copy of the task. The SW specifications are therefore smaller and less likely to contain errors.

Replicating tasks involves both creating copies of the task and resolving naming conflicts caused by the replication. Copying the task is simple; resolving the naming conflicts is more difficult. Name resolution involves processing each task in the workload. Any reference to the replicated task must be replaced with a reference to the appropriate copy of the task. SWSL defines rules for determining which copy to reference. These rules may be superseded in the specification of an individual component by explicitly specifying which copy is to be used.

## THE SYNTHETIC WORKLOAD

The output of the SWG is an executable SW. Our prototype SW is described in (Kiskis and Shin, 1990). The SW executes on a distributed system. Each processor executes a driver task and the appropriate SATs. The driver controls the activities of the SW in the context of the experiment. An experiment is divided into a number of independent runs. A *run* is a single execution cycle of the SW. During each run, the SW is initialized by the driver, and the SATs execute and eventually terminate. In the SWSL description of the SW, the user specifies the number of runs. For each component of the workload, different parameter values may be specified for each run.

At the beginning of each run, the driver initializes the SW. It reads the parameter tables which were produced by the

SWG and creates the specified SATs. Next, the drivers on all processors synchronize. Once synchronized, they begin the execution of their respective SATs as specified by the SATs' parameters for that run. By synchronizing at the beginning of each run, the driver ensures that the SW's behavior will stabilize quickly. The SW must be executing stably before accurate measurements may be made on the system. There are two ways to specify the end of a run in the SWSL specification. The first is to specify a time limit for the run. The second is to specify a condition, which, when met, indicates to the driver that the run has completed. An example of such a condition is the completion of $N$ executions of a specific periodic SAT. When the driver determines that a run is over, it stops the execution of the SW. All SATs are reset and system resources are returned to their initial states. The driver then waits before beginning the next run. This wait gives the user an opportunity to upload locally stored performance data or to reset external measurement devices. The driver begins the next run when it receives a signal from the user.

The SW is designed to be compatible with a wide range of performance measurement techniques. It executes as an application on the target system. Therefore, it may be used with any measurement mechanism which is part of the hardware, system software, or which is external to the system. It requires no special support and therefore will not interfere with these mechanisms. It also may be used with software measurement mechanisms which are not part of the system software. These measurement tasks may be specified as SATs. They will be invoked by the driver and will execute for the duration of the run.

## SUMMARY AND FUTURE WORK

As real-time systems become larger and more complex, we need more sophisticated tools to analyze their performance. The SWG suite is one such tool. It is designed to produce SWs which execute on distributed real-time systems. The workload model and corresponding language are specifically defined to describe the structure and behavior parameters of real-time workloads. The SWG supports features such as replication of tasks which facilitate its use on a distributed system. Finally, the SW is designed to support experimentation.

The SWG as described is operational. All functions described in this paper have been implemented. We will be using the SWG to make baseline performance measurements of the experimental, distributed real-time system HARTS and its operating system HARTOS. Both HARTS and HARTOS are under development at the Real-Time Computing Laboratory at the University of Michigan. As we use the SWG suite and become more experienced with the problems of performance evaluation, we will be upgrading the SWG software to incorporate new features.


Baird, R. (1973). APET - a versatile tool for estimating computer application performance. *Software - Practice and Experience, 3,* 385–395.

Bruyn, W., R. Jensen, D. Keskar, and P. Ward (1988). ESML: An extended systems modeling language based on the data flow diagram. *ACM Software Engineering Notes, 13,* 1, 58–67.

DeMarco, T. (1978). *Structured Analysis and System Specification.* Prentice-Hall, New Jersey.

Hatley, D. J., and I. A. Pribhai (1987) *Strategies for Real-Time System Specification.* Dorset House Publishing, New York.

Kiskis, D. L. and K. G. Shin (1990). A synthetic workload for real-time systems. In *Proc. Seventh IEEE Workshop on Real-Time Operating Systems and Software*, pp. 77–81.

Singh, A. (1981) *Pegasus: A controllable, interactive, workload generator for multiprocessors. Master's thesis*, Carnegie-Mellon University.

Singh, A., and Z. Segall (1982). Synthetic workload generation for experimentation with multiprocessors. In *Proc. Int'l Conf. on Distributed Computing Systems*, pp. 778–785.

Walters, R. E. (1976). Benchmark techniques: a constructive approach. *The Computer Journal, 19,* 1, 50–55. 2

Ward, P. T. (1986). The transformation schema: An extension of the data flow diagram to represent control and timing. *IEEE Trans. Software Engineering, SE-12,* 2, 198–210. 2

Ward, P. T., and S. J. Mellor (1986). *Structured Development for Real-Time System*, Vol. 1-3, Yourdon Press, Englewood Cliffs.